Лабораторна робота №4

Тема: Події браузера

Мета: Програмування реакції на події браузера

ТЕОРЕТИЧНІ ВІДОМОСТІ

Введення в браузерні події

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

Ось список найбільш часто використовуваних DOM-подій, поки просто для ознайомлення:

Події миші:

- `click` - відбувається, коли скликали на елемент лівою кнопкою миші (на пристроях з сенсорними екранами воно відбувається при торканні).
- `contextmenu` - відбувається, коли скликали на елемент правою кнопкою миші.
- `mouseover/mouseout` – коли миша наводиться на / покидає елемент.
- `mousedown/mouseup` - коли натиснули / відпустили кнопку миші на елементі.
- `mousemove` - при переміщеннні миші.

Події на елементах управління:

- `submit` - користувач відправив форму `<form>`.
- `focus` - користувач фокусується на елементі, наприклад натискає на `<input>`.

Клавіатурні події:

- `keydown i keyup` - коли користувач натискає / відпускає клавішу.

Події документа:

- `DOMContentLoaded` - коли HTML завантажений і оброблений, DOM документа повністю побудований і доступний.

CSS events:

- `transitionend` - коли CSS-анімація завершена.

Існує безліч інших подій. Ми детально розберемо їх в наступних розділах.

Обробники подій

Події можна призначити *обробник* , тобто функцію, яка спрацює, як тільки подія відбулася.

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

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

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

```
<input value="Нажми yf мене" onclick="alert('Клік!')"
type="button">
```

При натисканні мишкою на кнопці виконається код, вказаний в атрибуті onclick.

Зверніть увагу, для вмісту атрибута onclick використовуються одинарні лапки, так як сам атрибут знаходиться в подвійних. Якщо ми забудемо про це і поставимо подвійні лапки всередині атрибута, ось так: onclick="alert("Click!")", код не буде працювати.

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

Наступний приклад при натисканні запускає функцію countRabbits():

```
<script>
  function countRabbits() {
    for(let i=1; i<=3; i++) {
      alert("Кролик номер " + i);
    }
  }
</script>

<input type="button" onclick="countRabbits()"
value="Рахувати кроливів!">
```

Використання властивості DOM-об'єкта

Можна призначати обробник, використовуючи властивість DOM-елемента on<подія>.

Наприклад, elem.onclick:

```
<input id="elem" type="button" value="Нажми мене!">
<script>
elem.onclick = function() {
  alert('Спасибі');
};
</script>
```

Якщо обробник заданий через атрибут, то браузер читає HTML-розмітку, створює нову функцію з вмісту атрибута і записує в властивість.

Цей спосіб, по суті, аналогічний попередньому.

Оброблювач завжди зберігається у властивості DOM-об'єкта, а атрибут - лише один із способів його ініціалізації.

Ці два приклади коду працюють однаково:

Тільки HTML:

```
<input type="button" onclick="alert('Клік!')"
value="Кнопка">
```

HTML + JS:

```
<input type="button" id="button" value="Кнопка">

<script>

  button.onclick = function() {

    alert('Клік!');  };

</script>
```

Так як у елемента DOM може бути тільки одна властивість з ім'ям onclick, то призначити більше одного обробника так не можна.

У прикладі нижче призначення через JavaScript перезапише обробник з атрибута:

```
<input type="button" id="elem" onclick="alert('Було')"
value="Нажми мене">

<script>

  elem.onclick = function() { // перезапише обробник

    alert('Стане'); // виділяє тільки це

  };

</script>
```

До речі, оброблювачем можна призначити і вже існуючу функцію:

```
function sayThanks() {

  alert('Спасибі!');

}

elem.onclick = sayThanks;
```

Прибрати обробник можна призначенням `elem.onclick = null`.

Доступ до елементу через this

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

У коді нижче `button` виводить свій вміст, використовуючи `this.innerHTML`:

```
<button onclick="alert(this.innerHTML)">Нажми меня</button>
```

Використовуйте саме функції, а не рядки.

Призначення обробника рядком `elem.onclick = "alert(1)"` також спрацює. Це зроблено з міркувань сумісності, але робити так не рекомендується.

addEventListener

Фундаментальний недолік описаних вище способів призначення обробника - неможливість повісити кілька обробників на одну подію.

Наприклад, одна частина коду хоче при кліці на кнопку робити її підсвічується, а інша - видавати повідомлення.

Ми хочемо призначити два обробника для цього. Але нове DOM-властивість перезапише попереднє:

```
input.onclick = function() { alert(1); }

// ...

input.onclick = function() { alert(2); } // зміна попередн
обробник
```

Розробники стандартів досить давно це зрозуміли і запропонували альтернативний спосіб призначення обробників за допомогою спеціальних методів `addEventListener` і `removeEventListener`. Вони вільні від зазначеного недоліку.

Синтаксис додавання обробника:

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

event

Ім'я події, наприклад "click".

handler

Посилання на функцію-обробник.

options

Додатковий об'єкт з властивостями:

- once: Якщо true, тоді обробник буде автоматично видалений після виконання.
- capture: Фаза, на якій повинен спрацювати обробник. Так історично склалося, що options може бути false/true, це те ж саме, що {capture: false/true}.
- passive: Якщо true, то вказує, що обробник ніколи не викличе preventDefault().

Для видалення обробника слід використовувати removeEventListener:

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

Видалення вимагає саме ту ж функцію

Для видалення потрібно передати саме ту функцію-обробник яка була призначена.

```
function handler() {
  alert( 'Спасибо!' );
}
input.addEventListener("click", handler);
// ....
input.removeEventListener("click", handler);
```

Звернемо увагу - якщо функцію обробник не збереглися де-небудь, ми не зможемо її видалити. Немає методу, який дозволяє отримати з елемента обробники подій, призначені через addEventListener.

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

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

Як видно з прикладу вище, можна одночасно призначати обробники і через DOM-властивість і через addEventListener. Однак, щоб уникнути плутанини, рекомендується вибрати один спосіб.

Обробники деяких подій можна призначати тільки через addEventListener

Існують події, які не можна призначити через DOM-властивість, але можна через addEventListener.

Наприклад, таке подія DOMContentLoaded, яке спрацьовує, коли завершено завантаження і побудова DOM документа.

```
document.onDOMContentLoaded = function() {
```

```
  alert("DOM побудований"); // не працює
};
document.addEventListener("DOMContentLoaded", function() {
  alert("DOM побудований "); // а так працює
});
```

Так що `addEventListener` більш універсальний. Хоча зауважимо, що таких подій меншість, це швидше виняток, ніж правило.

Об'єкт події

Щоб добре обробити подія, можуть знадобитися деталі того, що сталося. Не просто «клік» або «натискання клавіші», а також - які координати покажчика миші, яка клавіша натиснута і так далі.

Коли відбувається подія, браузер створює *об'єкт події* , записує в нього деталі і передає його в якості аргументу функції-обробника.

Приклад нижче демонструє отримання координат миші з об'єкта події:

```
<input type="button" value="Нажми меня" id="elem">


<script>
  elem.onclick = function(event) {


    alert(event.type + " на " + event.currentTarget);
    alert("Координати: " + event.clientX + ":" +
event.clientY);
  };
</script>
```

Деякі властивості об'єкта event:

`event.type` Тип події, в даному випадку "click".

`event.currentTarget` Елемент, на якому спрацював обробник. Значення - зазвичай таке ж, як і у `this`, але якщо обробник є функцією-стрілкою або за допомогою `bind` прив'язаний інший об'єкт в якості `this`, то ми можемо отримати елемент з `event.currentTarget`.

`event.clientX / event.clientY`

Координати курсора в момент кліка щодо вікна, для подій миші.

Є також і ряд інших властивостей, в залежності від типу подій, які ми розберемо в подальших главах.

Об'єкт події доступний і в HTML

При призначенні обробника в HTML, теж можна використовувати об'єкт `event`, ось так:

```
<input type="button" onclick="alert(event.type)" value="Тип події">
```

Це можливо тому, що коли браузер з атрибута створює функцію-обробник, то вона виглядає так: `function(event) { alert(event.type) }`. Тобто, її перший аргумент називається `"event"`, а тіло взято з атрибута.

ОТЖЕ

Є три способи призначення обробників подій:

1. Атрибут HTML: **`onclick="..."`**.
2. DOM-властивість: **`elem.onclick = function`**.
3. Спеціальні методи: `elem.addEventListener(event, handler[, phase])` для додавання, **`removeEventListener`** для видалення.

ЗАВДАННЯ ДЛЯ САМОСТІЙНОГО ВИКОНАННЯ

1. Дано картинки. Прив'яжіть до кожної картинки подія, щоб при натисканні на картинку alert виводився її src.
2. Дано два посилання. Прив'яжіть всіх посиланнь подію: з наведення на посилання в атрибут title запишеться її текст.
3. Прив'яжіть до всіх посиланнях подію: з наведення на посилання в кінець її тексту дописується її href в круглих дужках.
4. Доповніть попередню задачу: після першого наведення на посилання слід відв'язати від неї подію, яка додає href в кінець тексту.
5. Прив'яжіть всім input таку обставину: з втратою фокусу кожен input виводить своє value в абзац з id = "test".
6. Для всіх input зробіть так, щоб вони виводили свій value через Alert при натисканні на будь-який з них, але тільки за першим натискання. Повторне натискання на інпут не повинно викликати реакції.
7. Дано абзаци з числами. При натисканні на абзац в ньому повинен з'явиться квадрат числа, яке він містить.
8. Дано input. Зробіть так, щоб всі input з втратою фокусу перевіряли свій вміст на правильну кількість символів. Скільки символів має бути в input, вказується в атрибуті data-length . Якщо вбито правильну кількість, то межа input стає зеленою, якщо неправильне - червоною.
9. Додайте JavaScript до кнопки button, щоб при натисканні елемент зникав.
10. Створіть кнопку, яка буде приховувати себе після натискання.
11. Створити меню, яке після натискання відкривається або закривається.
12. Створіть «Карусель» - стрічку зображень, яку можна гортати вліво-вправо натисненням на стрілки.