Temat 17

Temat: Zdarzenia

Obiektowy model dokumentu HTML DOM umożliwia reagowanie poprzez język JavaScript na zdarzenia powstałe w przeglądarce internetowej głównie na skutek interakcji z użytkownikiem.

Zdarzenia to czynności, które dzieją się w przeglądarce. Może je wywoływać użytkownik, ale i każdy element na stronie. Przykładowo klikając na przycisk na stronie, wywołujemy zdarzenie click. Wybierając za pomocą klawisza tab kolejny element w formularzu, wywołujemy zdarzenie focus. Opuszczając taki element, wywołujemy blur. Obrazek się wgrał? Wywołuje zdarzenie load. Formularz się wysyła? Wywoływane jest zdarzenie submit. Takich zdarzeń jest oczywiście o wiele, wiele więcej.

Typ zdarzenia:	**Wydarzenie jest uruchamiane ...**
click	po naciśnięciu i zwolnieniu głównego przycisku myszy, trackpada itp.
dblclick	gdy podwójnie klikniemy na obiekt (np. input)
change	gdy obiekt zmienił swoją zawartość (np. pole tekstowe)
mouseover	po przesunięciu kursora myszy nad elementem.
mouseout	gdy kursor opuścił obiekt
mousemove	gdy kursor myszy został przesunięty
submit	gdy formularz został wysłany
resize	gdy rozmiar okna przeglądarki jest zmieniany
focus	gdy obiekt stał się wybrany (np. pole tekstowe)
blur	gdy obiekt przestał być aktywny (np. input)
keydown	gdy został naciśnięty klawisz na klawiaturze
keyup	gdy puścimy klawisz na klawiaturze
input	podobne do powyższego, ale odpalane synchronicznie w czasie trzymania klawisza (np. przytrzymanie klawisza A w polu tekstowym)
load	gdy obiekt został załadowany (np. cała strona)
contextmenu	gdy kliknięto prawym klawiszem myszki i pojawiło się menu kontekstowe
wheel	gry kręcimy kółeczkiem myszki

select	gdy zawartość obiektu została zaznaczona
unload	gdy użytkownik opuszcza dana stronę
DOMContentLoaded	po pełnym załadowaniu dokumentu DOM

Powyższa lista zawiera tylko częściej używane zdarzenia, jest ich o wiele więcej. Możesz je znaleźć np. na stronie: https://www.w3schools.com/jsref/dom_obj_event.asp

Obsługa zdarzeń

Istnieją 3 sposoby przypisania funkcji obsługi zdarzeń:

1. Atrybut HTML: onclick="...".
2. Obiekt DOM: elem.onclick = function.
3. Metody:
   - elem.addEventListener(event,handler[,phase])– dodawanie zdarzenia,
   - removeEventListener - usuwanie zdarzenia.

Atrybut HTML - Obsługa zdarzeń w kodzie HTML

Program obsługi można ustawić w kodzie HTML z atrybutem o nazwie **`on<event>`.**

Na przykład, aby przypisać procedurę **`click`** obsługi dla input, możemy użyć onclick.

```
<input type="button" onclick="alert('Click!')" value="Click me">
```

Po kliknięciu myszą, kod wewnątrz **`onclick`** zadziała.

Należy pamiętać, że w środku **`onclick`** stosujemy apostrofy, ponieważ sam atrybut występuje w podwójnych cudzysłowach. Jeśli zapomnimy, że kod znajduje się wewnątrz atrybutu i użyjemy podwójnych cudzysłowów w środku, w ten sposób **`onclick="alert("Click!")"`**, kod nie będzie działać poprawnie.

Atrybut HTML nie jest wygodnym miejscem do napisania dużej ilości kodu, więc lepiej stworzyć funkcję JavaScript podpiętą do zdarzenia. Znasz już ten sposób wykorzystania zdarzenia click.

Metoda obecnie nie jest polecana, gdyż mieszamy tutaj dwa języki: język strony HTML i język JavaScript i to w jednej linijce. Jest to rozwiązanie niezbyt eleganckie, mało czytelne, jednak działające i jeszcze do niedawna bardzo często stosowane. Przy wielości tego typu elementów poprawianie kodu strony może przysporzyć wiele problemów, nie mówiąc już o możliwości przechowywania kodu skryptu w odrębnym pliku.

Przykład 1. Przygotuj stronę html wykorzystującą poniższy kod

```
<body>
    <input type="button" onclick="countRabbits()" value="Policz króliki!">
    <script>
        function countRabbits() {
            for(let i=1; i<=3; i++) {
                alert("Królik numer " + i);
            }
        }
    </script>
</body>
```

Właściwość DOM - Obsługa zdarzeń jako właściwość obiektu

Jest to starsza metoda obsługi zdarzeń. Polega na przypisaniu do danego zdarzenia własnej funkcji lub funkcji anonimowej jako właściwości danego obiektu.

Pamiętaj, że aby mieć pewność, że kod nie odwołuje sie do obiektów, które jeszcze nie zostały wczytane do pamięci przeglądarki, **kod obsługi zdarzeń najlepiej wstawiać na końcu kodu strony lub umieszczać go w module.**

Możemy przypisać procedurę obsługi zdarzenia za pomocą właściwości DOM **`on<event>`**.

Przykład 2. Przygotuj stronę html wykorzystującą poniższy kod

```
6     <style>
7     body{
8         background-color: #002255;
9         color: #fff;
10        font-size: 20px;
11    }
12    .module{
13    background-color: #009;
14    height:150px;
15    width:150px;
16    border: 1px solid #fff;
17    margin:10px 0px;
18    }
19    .module1{
20    background-color: #090;
21    height:100px;
22    width:100px;
23    border: 1px solid #fff;
24    margin:50px 50px;
25    }
26    </style>
27</head>
28<body>
29    <input id="elem" type="button" value="Kliknij mnie">
30    <div class="module"></div>
31    <script>
32        elem.onclick = function FunClick() {
33            alert('Dziekuję');
34        };
35        const div= document.querySelector(".module");
36        div.onclick= function FunClick2() {
37            div.classList.toggle("module1");
38        };
39    </script>
```

Możemy też przypisać istniejącą funkcję bezpośrednio do obsługi:

Przykład 3. Przygotuj stronę html wykorzystującą poniższy kod

```
<body>
    <input id="elem" type="button" value="Kliknij mnie">
    <div class="module"></div>
    <script>
    // tworzymy funkcje
    function FunClick() {
        alert('Dziekuję');
    };
    function FunClick2() {
            div.classList.toggle("module1");
        };
    //podpinamy funkcje do zdarzeń
    elem.onclick =FunClick; //nazwa funkcji bez nawiasów!!
    const div= document.querySelector(".module");
    div.onclick= FunClick2;
    </script>
</body>
```

Uwaga: Funkcja przy podpięciu powinna być przypisana jako `FunClick` a nie `FunClick()!!!`. Dzieje się tak, ponieważ nie chcemy odpalać funkcji, a tylko ją podpiąć pod dane zdarzenie.

Przypisanie zdarzenia do wielu obiektów:

Przykład 4. Przygotuj stronę html wykorzystującą poniższy kod

```
    <style>
    body{
        background-color: #002255;
        color: #fff;
        font-size: 20px;
    }
    .mark{
    background-color: #009;
    height:100px;
    border: 1px solid #fff;
    margin:10px 0px;
    text-align:center;
    }
    </style>
</head>
<body>
    <p>Akapit 1</p>
    <p>Akapit 2</p>
    <p>Akapit 3</p>
    <p>Akapit 4</p>
    <script>
    const p = document.querySelectorAll('p');
    for (let i=0; i<p.length; i++) {
        p[i].onclick = function () {
            this.classList.toggle('mark');
        }
    }
    </script>
```

Aby **usunąć wcześniej przypisane zdarzenie**, wystarczy pod daną właściwość podstawić null:

```
elem.onclick = null
```

W powyższych sposobach przypisywania procedur obsługi **nie możemy przypisać wielu procedur obsługi do jednego zdarzenia**. Gdy chcemy przypisać do elementu dwa programy obsługi zdarzeń w powyższych sposobach, nowa właściwość DOM nadpisze istniejącą:

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

input.onclick = function() { alert(2); }
```

Aby osiągnąć przypisanie wielu działań możliwe jest zastosowanie konstrukcji:

```
input.onclick = function() {
      alert(1);
      alert(2);
      print();
}
```

Nie ma jednak wtedy łatwej metody na usuwanie i odpisanie pojedynczych funkcji.

Metody

Obecnie zaleca się zarządzanie zdarzeniami za pomocą specjalnych metod **`addEventListener i removeEventListener`**. Są one wolne od powyższego problemu.

Składnia wyjściowa:

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

event - Nazwa zdarzenia, np "click".
handler - Funkcja obsługi.
phase - Opcjonalny argument, który włącza lub wyłącza (true/false) propagację zdarzeń.

Aby usunąć program obsługi, użyj **`removeEventListener`** (przyjmuje ona 2 obowiązkowe parametry: nazwę zdarzenia i **nazwę funkcji** którą chcemy wyrejestrować oraz parametr opcjonalny):

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

Przykład 5. Przygotuj stronę html wykorzystującą poniższy kod

Po pojedynczym kliknięciu zostaną wywołane obie funkcje.

```
    body{
        background-color: #002255;
        color: #fff;
        font-size: 20px;
    }
    .mark{
    background-color: #009;
    height:100px;
    border: 1px solid #fff;
    margin:10px 0px;
    text-align:center;
    }
    </style>
</head>
<body>
    <div class="test">
    </div>
    <input id="elem" type="button" value="Kliknij mnie">
    <script>
      function handler1() {
        alert('Dziękuję!');
      };
      function handler2() {
            const   el = document.createElement("div");
            el.id = "myDiv";
            el.innerText = "Tekst w divie";
            el.classList.add("mark");
            const div = document.querySelector(".test");
            div.appendChild(el);
            }
      elem.addEventListener("click", handler1);
      elem.addEventListener("click", handler2);
    </script>
```

Nasłuchiwanie zdarzeń

Aby zareagować na wykonanie zdarzenia, powinniśmy podpiąć dla danego elementu funkcję nasłuchującą, która zostanie odpalona w momencie wykonania danego zdarzenia.

Przykład 6. Przygotuj stronę html wykorzystującą poniższy kod

```
<body>
    <input type="button" id="hider" value="Schowaj tekst">
    <input type="button" id="hider1" value="Pokaż tekst">
    <div id="text">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer purus
    magna, dignissim convallis dictum quis, maximus eget arcu. Morbi eget convallis est. Ut
    dictum interdum risus, nec imperdiet magna feugiat a. Aliquam eu erat ante. Pellentesque
    gravida, nisl et maximus varius, ante ligula bibendum sem, eleifend interdum nibh nisi
    vitae sem. Ut placerat, diam vel fringilla varius, sem sapien ultricies elit, at feugiat
    ante diam in purus. Sed varius ultricies elit at convallis. Aliquam congue dapibus nulla
    convallis ornare. Nullam ut faucibus massa, eget laoreet neque.</div>
  <script>
    // zdarzenie jako właściwość
    document.getElementById('hider').onclick = function() {
    document.getElementById('text').style.display="none";
    }
    //zdarzenie dodane funkcją addEventListener
    function handler1() {
        document.getElementById('text').style.display="block";
      };
    const div = document.querySelector("#hider1");
    hider1.addEventListener("click", handler1);
    </script>
</body>
```

Przykład 7. Przygotuj stronę html wykorzystującą poniższy kod

```
<body>
    <input type="button" id="hider" value="Schowaj tekst">
    <input type="button" id="hider1" value="Pokaż tekst">
    <input type="button" id="hider2" value="Zmień tekst">
    <div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer purus magna,
    dignissim convallis dictum quis, maximus eget arcu. Morbi eget convallis est. Ut
    dictum interdum risus, nec imperdiet magna feugiat a. Aliquam eu erat ante.
    Pellentesque gravida, nisl et maximus varius, ante ligula bibendum sem, eleifend
    interdum nibh nisi vitae sem. Ut placerat, diam vel fringilla varius, sem sapien
    ultricies elit, at feugiat ante diam in purus. Sed varius ultricies elit at
    convallis. Aliquam congue dapibus nulla convallis ornare. Nullam ut faucibus massa,
    eget laoreet neque.</div>
  <script>
    const event1 = document.querySelector("#hider");
    const event2 = document.querySelector("#hider1");
    const event3 = document.querySelector("#hider2");
    const text1 = document.querySelector("div");

    //możemy też podpiąć klasyczną funkcję anonimową
    event1.addEventListener("click", function(){
        text1.style.display="none";
        });
    //lub funkcję strzałkową
    event2.addEventListener("click", () => {
        text1.style.display="block";
        });
    //lub funkcję strzałkową z parametrem
    event3.addEventListener("click", e => {
        text1.classList.toggle('fun');
        console.log(e);
        });
    </script>
</body>
```

Przykład 8. Przygotuj stronę html wykorzystującą poniższy kod

```
<body>
        <div id="myDiv" >Moje pudełeczko</div>
        <p class="output"></p>
  <script>
        const button = document.querySelector("#myDiv");
        const inf = document.querySelector(".output");
        button.addEventListener("mouseover", hovered, false);
        button.addEventListener("mouseout", hoveredOut, false);

        function hovered(e) {
            document.querySelector("#myDiv").style.backgroundColor="red";
            inf.innerText = "Myszka w środku!";
        }
        function hoveredOut(e) {
            document.querySelector("#myDiv").style.backgroundColor="green";
            inf.innerText ="Myszka na zewnątrz!";
        }
    </script>
</body>
```

Aby wyrejestrować zdarzenie można użyć metody:

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

UWAGA - jeśli nie przechowujemy funkcji w zmiennej lub używamy funkcji anonimowych, nie możemy usunąć zdarzenia zarejestrowanego przy pomocy **`addEventListener`**.

Przykład 9. Przygotuj stronę html wykorzystującą poniższy kod

```
<body>
    <button class="button">Odlicz</button>
    <script>
        const btn = document.querySelector(".button");
        let counter = 0;

        function elementClick() {
            counter++;
            btn.innerText = `Klik numer ${counter}`;
            if (counter >= 5) {
                btn.removeEventListener("click", elementClick);
            }
        }
        btn.addEventListener("click", elementClick);
    </script>
</body>
```

DOMContentLoaded

Ważną częścią pracy z JavaScriptem jest zapewnienie, by kod działał we właściwym czasie. Często nie wystarczy umieszczenie kodu u dołu strony i oczekiwanie, że wszystko zadziała po załadowaniu strony. W wielu sytuacjach może być konieczne dodanie dodatkowego elementu kodu tak, aby kod nie działał, zanim strona nie będzie gotowa.

Aby podpięcie zdarzenia do DOM mogło zadziałać, elementy muszą być już dostępne dla skryptu. Oznacza to, że zanim zaczniemy cokolwiek podpinać, musimy się upewnić, że został już wczytany html i zostało stworzone drzewo dokumentu.

Aby wykryć czy dokument został wczytany, możemy skorzystać ze zdarzenia **DOMContentLoaded** (UWAGA: działa tylko z **addEventListener)**:

```
document.addEventListener("DOMContentLoaded", function(event) {
    console.log("DOM ");
    ...
});
```

Przykład10. Przetestuj działanie skryptu. Dodaj kilka elementów do listy, zamknij ją. Spróbuj ponownie dodać elementy listy.

```
<body>
    <h1 id="theTitle">Lista zakupów</h1>
    <button id="add">Dodaj pozycję do listy</button>
    <button id="close">Zamknij listę</button>
    <ol id="list">
      <li>pieczywo</li>
      <li>mleko</li>
      <li>ser</li>
    </ol>
    <script>
    window.addEventListener("DOMContentLoaded", init, false);
    function init(){
        // funkcja dodająca element listy
        function add() {
            let newLi = document.createElement('li');
            let tekst= prompt("Podaj co jeszcze kupić")
            newLi.innerText = tekst;
            const list= document.querySelector("#list");
            list.insertBefore(newLi, list.children[1]);
            }
        // funkcja wyłączająca zdarzenie click -add
        function close() {
            const button1 = document.querySelector("#add");
            button1.removeEventListener("click", add);
            }
        // podpięcie do przycisku 1 zdarzenia
        const button1 = document.querySelector("#add");
        button1.addEventListener("click", add);
        // podpięcie do przycisku 2 zdarzenia
        const button2 = document.querySelector("#close");
        button2.addEventListener("click", close);
    }
    </script>
</body>
```

W wielu skryptach zamiast **`DOMContentLoaded`** używane jest zdarzenie **`load`** dla obiektu **`window`.** Event **`load`** dla **`window`** jest odpalany, gdy wszystkie elementy na stronie zostaną załadowane - nie tylko drzewo dom, ale także grafiki. Bardzo często będzie to powodować mocno zauważalne opóźnienia, gdy np. dynamicznie generowane za pomocą JS menu odpali się dopiero po wczytaniu dużych grafik. Jeżeli więc skrypt ma działać tylko na elementach, a nie czekać na wczytanie całych grafik, użyj zdarzenia **`DOMContentLoaded`**.

Własności obiektu event

Możemy pobrać dane na temat wybranego zdarzenia. W przypadku klawisza możemy wykryć, który dokładnie został naciśnięty. Po kliknięciu (najechaniu itp.) w dane miejsce możemy pobrać dokładne koordynaty (współrzędne), gdzie na ekranie był kursor, gdy to zdarzenie nastąpiło. W tym celu możemy skorzystać z właściwości obiektu event:

event.button - Zwraca przycisk myszy.

event.clientX - Zwraca poziomą pozycję zdarzenia w obszarze klienta.

event.clientY -Zwraca pionową pozycję zdarzenia w obszarze klienta.

***clientX* oraz *clientY* –** to współrzędne względne kursora myszy (względem elementu który dostarczył procedurę obsługi zdarzenia),

event.ctrlKey -Zwraca wartość logiczną wskazującą, czy klawisz <ctrl> był wciśnięty podczas zdarzenia.

event.keyCode -Zwraca kod Unicode dla klawisza nie będącego znakiem w zdarzeniu keypress lub dowolnego klawisza w każdym innym zdarzeniu związanym z klawiaturą.

event.pageX -Zwraca poziomą współrzędną miejsca, gdzie wystąpiło zdarzenie, względem całej strony.

event.pageY -Zwraca pionową współrzędną miejsca, gdzie wystąpiło zdarzenie, względem całej strony.

event.offsetX- Zwraca współrzędną poziomą wskaźnika myszy względem położenia krawędzi elementu docelowego

event.offsetY- Zwraca współrzędną pionową wskaźnika myszy względem położenia krawędzi elementu docelowego

event.type -Zwraca nazwę zdarzenia.

Przykład 11. Włącz konsolę i przetestuj działanie skryptu

```
<head>
    <title></title>
    <meta charset="UTF-8">
    <title>Zdarzenia  - T17p11</title>
    <style>
    section{
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
        }
    .card{
        width: 600px;
        height:100px;
        background-color: #ddd;
    }
    </style>
</head>
```

```
<body>
    <section>
        <h2> </h2>
        <div class="card">
            <h3> Uruchom konsolę i kliknij kilkukrotnie w polu</h3>
        </div>
        <p>Naciśnij kilka klawiszy na klawiaturze w polu wprowadzania.</p>
        <input type="text">
        <p id="demo"></p>
    </section>
<script>
    const card = document.querySelector('.card');
    const heading = document.querySelector('h2');
    const inputT = document.querySelector('input');
    // Click
    card.addEventListener('click', runEvent);
    //keydown
    inputT.addEventListener('keydown', runEvent2);

    // Event Handler
    function runEvent(e) {
        console.log("EVENT TYPE: "+e.type);
        heading.textContent = `clientX: ${e.clientX}; clientY: ${e.clientY}`;
        const x = e.pageX;
        const y = e.pageY;
        console.log(x,y);
    }
    function runEvent2(event) {
        console.log("EVENT TYPE: "+event.type);
        const x = event.key;
        document.querySelector("#demo").textContent = `Wciśnięto klawisz: ${x}`;
    }
</script>
</body>
```