Technologie Internetu. Zdarzenia

Aleksander Denisiuk (denisjuk@pja.edu.pl)
Polsko-Japońska Akademia Technik Komputerowych
Wydział Informatyki w Gdańsku
ul. Brzegi 55, 80-045 Gdańsk

27 maja 2020

Zdarzenia

Najnowsza wersja tego dokumentu dostępna jest pod adresem
http://users.pja.edu.pl/~denisjuk/

Podstawy zdarzeń

Zdarzenia

- ✔ Prawie wszystkie JavaScript aplikacje reagują na *zdarzenia*
- ✔ Rodzaje zdarzeń:
  - ✘ Zdarzenia myszy (`click`, `contexrmenu`, `moseover`, ...)
  - ✘ Zdarzenia elementów sterowania (`submit`, `focus`, ...)
  - ✘ Zdarzenia klawiatury (`keyup`, `keydown`, ...)
  - ✘ Inne (`load`, `DOMContentLoaded`, `transitionend` ...)

Aktywacja zdarzeń

- ✔ Żeby funkcja reagowała na zdarzenia, powinna ona zostać przypisana do tego zdarzenia
- ✔ Funkcja, reagująca na zdarzenie, nawysa się *callback*
- ✔ Są różne sposoby aktywacji funkcji zdarzenia

Atrybut emelentu


```
<input id="b1"
  value="Kliknij mnie"
  onclick="alert('Thanks!');"
  type="button" />
```

✔ Demo

Atrybut emelentu prim


```
<head>
  <script>
  function liczDoTrzech() {
    for(var i=1; i<=3; i++) {
       alert("licz do trzech: "+i)
    }
  }
  </script>
</head>
<body>
 <button onclick="liczDoTrzech()">
    Licz!
  </button>
</body>
```

✔ Demo

Atrybut emelentu. Zalety (i wady)

- ✔ Sposób jest prosty i czytelny
- ✔ Często jest używany
- ✔ Wszędzie działa
- ✔ Źle nadaje się dla skomplikowanych funkcji

Właściwość elementu DOM

- ✔ Rejestracja funkcji reagującej przez właściwość (nie atrybut!) elementu on*nazwa_zdarzenia*

```
<input id="El"
  value="Kliknij mnie"
  type="button" />
<script type="text/javascript">
document.getElementById('El').onclick
  = function() {
      alert('Thanks ;)')
    }
</script>
```

- ✔ Demo

Właściwość elementu. Uwagi

- ✔ Właściwość, nie atrybut: małe litery
- ✔ Callback jest *funkcją*, a nie tekstem
- ✔ Jeżeli przeglądarka widzi atrybut `onzdarzenie`, ona tworzy funkcję anonimową z zawartości atrybutu
- ✔ Wywołania `getElementById` powinny nastąpić po tym, jak element będzie istnieć. Najlepiej po załadowaniu okna.
- ✔ Można wyrejestrować callback

```
onzdarzenie=null
```

Właściwość elementu. Inne uwagi

- ✔ Można użyć nie anonimowej funkcji `document.getElementById('button').onclick = doSomething`
- ✔ W poprzednim przykładzie używamy `doSomething`, a nie *doSomething()*. W znaczniku elementu używamy

```
<tag onclick='doSmething()' />
```

  - ✘ Bo przeglądarka robi z atrybutu funkcję anonimową:

```
tag.onclick=function(){doSmething()}
```

Właściwość elementu. Zalety (i wady)

- ✔ Jest prostym sposobem
- ✔ Działa we wszystkich przeglądarkach
- ✔ Można zarejestrować tylko jeden callback
- ✔ Nie na każde zdarzenie można zarejestrować callback (na `transitionend` — nie można)

Dostęp do elementu

- ✔ `this` w funkcji jest odwoływaniem do elementu

```
<button onclick="alert(this.innerHTML)">
  Click me
</button>
```

- ✔ Demo

addEventListener, removeEventListener

- ✔ rejestracja funkcji

  `addEventListener(zdarzenie, funkcja, [faza])`
- ✔ wyrejestracja funkcji

  `removeEventListener(zdarzenie, funkcja, [faza])`
- ✔ przykład (domyślnie `faza=false`):

```
input.addEventListener("click", handler)
// ....
input.removeEventListener("click", handler)
```

addEventListener — Uwagi

- ✔ Nie działa w IE8- (`attachEvent/detachEvent`)
- ✔ Można zarejestrować kilka fukcji
- ✔ Nie można wyrejestrować funkcji anonimowej
- ✔ Wyrejestrowanie wymaga tej samej funkcji, następujący kod nie działa:

```
elem.addEventListener( "click",
                       function() {alert('Thanks')});
// ....
elem.removeEventListener( "click",
                       function() {alert('Thanks')});
```

✔ Przykłady

```
$(document).ready();

$("p").click(function(){
  alert(1);
});

$("p").dblclick(function(){
    $(this).hide();
});

$("p").on("click", function(){
    $(this).hide();
});
```

Kolejność zdarzeń

Potok główny

- ✔ Każde okno ma jeden główny wątek:
  - ✘ interpretuje JavaScript oraz wyświetla drzewo DOM
  - ✘ wykonuje polecenia szeregowo
  - ✘ jest blokowany przy wyświetleniu modalnych okien (np `alert`)

Potoki i procesy

- ✔ Potoki:
  - ✘ są równoległe wątki wewnętrzne (na przykład, sieciowe)
  - ✘ ładowanie danych nie jest blokowane przez `alert`
  - ✘ nie mamy na nie wpływu
- ✔ Web workers:
  - ✘ można uruchamiać dodatkowe *procesy*, Web Workers
  - ✘ komunikują z potokiem głównym
  - ✘ są rozdzielone i nie mają dostępu do DOM
  - ✘ wykorzystywane są głównie do obliczeń równoległych

Zdarzenia asynchroniczne

- ✔ Większość zdarzeń jest asynchroniczna:
  - ✘ w momencie zajścia dodaje się do kolejki potoku głównego
  - ✘ jest opracowywana szeregowo
  - ✘ Przykład: kliknięcie myszką (trzy zdarzenia)

Zdarzenia synchroniczne

- ✔ Część zdarzeń jest *synchroniczna* (zazwyczaj wywołane w jawny sposób metodą elementu)
- ✔ Opracowywana jest natychmiast

Zdarzenie synchroniczne: przykład


```
<input type="button" value="Kliknij mnie">
<input type="text" size="60">
<script>
  var button = document.body.children[0];
  var text = document.body.children[1];
  button.onclick = function() {
    text.value += ' -> click ';
    text.focus(); // zdarzenie focus
    text.value += ' click-> ';
  };
  text.onfocus = function() {
    text.value += ' !focus! ';
  };
</script>
```

✔ Zobacz

Zrobić zdarzenie asynchronicznym

- ✔ Aby przywrócić kolejność zdarzeń, można skorzystać z `setTimeout(func,0)`
  - ✘ funkcja `func` zostanie wywołana po zakończeniu bieżącej funkcji
  - ✘ a więc zdarzenie `focus` zostanie umieszczone w kolejce po bieżącym zdarzeniu

```
button.onclick = function() {
  ...
  setTimeout(function() {
    text.focus();
  }, 0);
  ...
};
```

- ✔ Zobacz

Przykład

- ✔ Zamienić wprowadzane litery na duże
- ✔ Opracować `keydown`?

```
<input id="my" type="text">
<script>
document.getElementById('my').onkeydown
 = function() {
   this.value = this.value.toUpperCase();
};
</script>
```

- ✔ Zobacz
- ✔ czemu tak jest?

Przykład

- ✔ Zdarzenie zachodzi po działaniu przeglądarki
- ✔ Opracować `keyup`?

```
<input id="my" type="text">
<script>
document.getElementById('my').onkeyup
 = function() {
   this.value = this.value.toUpperCase();
};
</script>
```

- ✔ Zobacz

Przykład

- ✔ Optymalnie: zastosować `setTimeout`!

```
<input id="my" type="text">
<script>
document.getElementById('my').onkeydown
  = function() {
    var self = this;
    function handle() {
      self.value = self.value.toUpperCase()
    }
    setTimeout(handle, 0);
 };
</script>
```

- ✔ Zobacz

Obiekt zdarzenia

Obiekt zdarzenia

- ✔ Obiekt zdarzenia zawiera całkowitą informację o zdarzeniu
  - ✘ element, na którym zaszło zdarzenie
  - ✘ współrzędne wskaźnika myszy
  - ✘ kod naciśniętego klawisza
  - ✘ etc

Obiekt zdarzenia

- ✔ Obiekt `event` jest pierwszym argumentem callbacka

```
function doSomething(event) {
    // event - obiekt zdarzenia
}

element.onclick = doSomething
```

Właściwości obiektu zdarzenia

- ✔ `event.type` typ zdarzenia (np. `click`)
- ✔ `event.currentTarget` element, na którym zareagował callback
- ✔ `event.clientX`/`event.clientY` współrzędne kursora myszki w momencie kliknięcia
- ✔ Inne zdarzenia mają inne specyficzne właściwości

W HTML

✔ Obiekt zdarzenia jest dostępny w HTML, na przykład:

```
<input type="button"
    onclick="alert(event.type)"
    value="Typ zdarzenia">
```

Propagacja i przechwytywanie

Przykład

- ✔ przy kliknięciu na włożonym elemencie działa funkcja, zarejestrowana na elemencie rodzicielskim
- ✔ zdarzenie *bąbelkuje* (propagacja, bubbling)

```
<div onclick="alert('Div!')">
  <em>Kliknij na <code>EM</code>,
  zobaczysz <code>DIV</code></em>
</div>
```

- ✔ Demo

Kolejność zdarzeń przy bąbelkowaniu

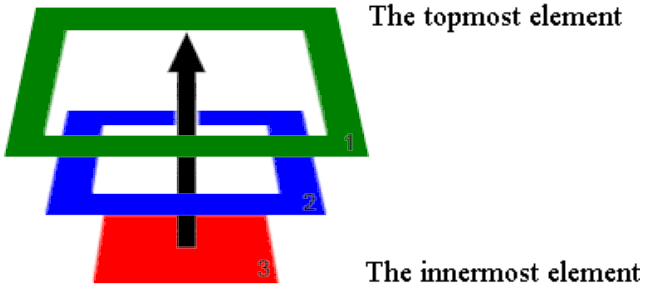
- ✔ Niech dane będą włożone elementy

```
<div class="d1" onclick="alert(1)">1
 <div class="d2" onclick="alert(2)">2
   <div class="d3" onclick="alert(3)">3</div>
 </div>
</div>
```

- ✔ Która z trzech funkcji reagujących powinna zadziałać przy kliknięciu na `d3`?
- ✔ Jeżeli wszystkie funkcje, to w jakiej kolejności?

Propagacja (bąbelkowanie, bubbling)

- ✔ Najpierw zdarzenie na wewnętrznym elemencie, potem poziom wyżej, etc



- ✔

```
<div class="d1" onclick="alert(1)">1
 <div class="d2" onclick="alert(2)">2
   <div class="d3" onclick="alert(3)">3</div>
 </div>
</div>
```

- ✔ Demo

Element bieżący

- ✔ `event.currentTarget(== this)` jest elementem, na którym została zarejesrtrowana odpalona funkcja

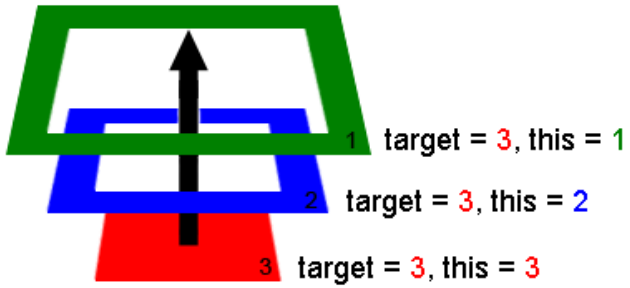
```
<div class="d1" onclick="highlight(this)">1
 <div class="d2" onclick="highlight(this)">2
  <div class="d3" onclick="highlight(this)">3
  </div>
 </div>
</div>

function highlight(elem) {
    elem.style.backgroundColor = 'yellow';
    alert(elem.className);
    elem.style.backgroundColor = '';
}
```

- ✔ Demo

Element docelowy `event.target`

- ✔ Najgłębszy element, na którym zaszło zdarzenie

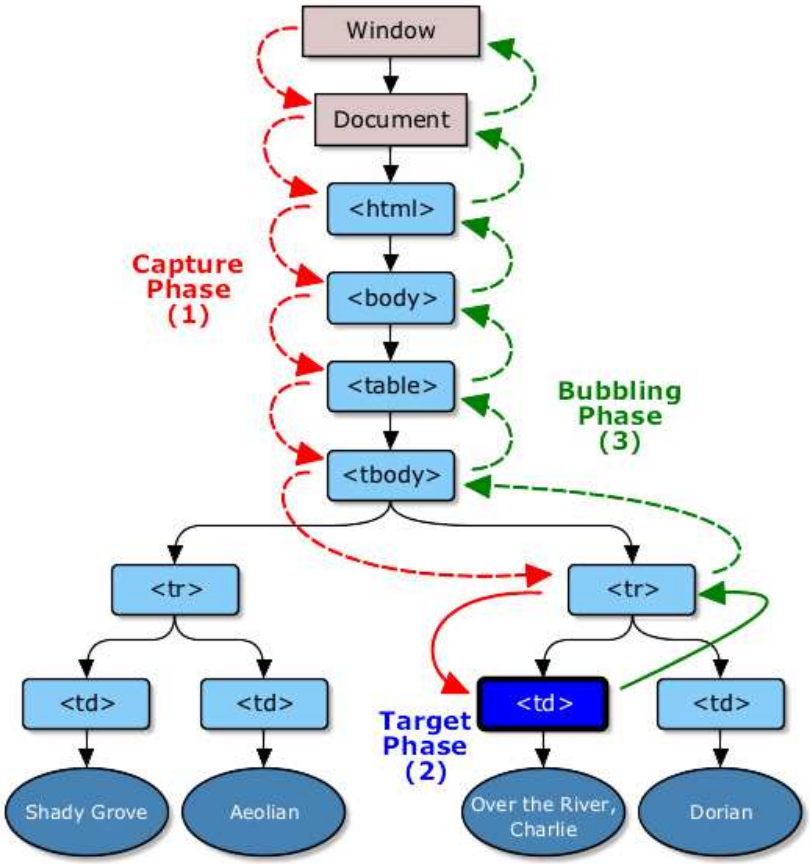


- ✔
- ✔ Demo

Zatrzymanie propagacji

- ✔ `event.stopPropagation()`
- ✔ Uwagi:
  - ✘ należy unikać
  - ✘ `event.stopImmediatePropagation()` zatrzymuje wszystkie callbacki na bieżącym elemencie

Trzy fazy zdarzenia



Trzy fazy

- ✔ przechwytywanie

  `addEventListener(zdarzenie, funkcja, true)`

  (pierwsza faza, capturing)
- ✔ Propagacja

  `addEventListener(zdarzenie,funkcja,false)`

  (trzecia faza, bubbling)
- ✔ na drugiej fazie (target) odpala się obydwie funkcje
- ✔ w pozostałych sposobach — rejestracja na fazę propagacji
- ✔ numer fazy: `event.eventPhase`
- ✔ Demo

Delegowanie zdarzeń

Delegowanie zdarzeń

- ✔ Mamy dużo elementów, które są opracowywane w podobny sposób
- ✔ Rejestrujemy funkcję reagującą na elemencie rodzicielskim i korzystamy z `event.target`
- ✔ Przykład:

```
<ul id="menu">
  <li><a href="/php">PHP</a></li>
  <li><a href="/html">HTML</a></li>
  <li><a href="/javascript">JavaScript</a>
                                        </li>
  <li><a href="/flash">Flash</a></li>
</ul>
```

Delegowanie. JavaScript


```
document.getElementById('menu').onclick
        = function(event) {
  var target = event.target;

  if (target.tagName != 'A') return;

  var href = target.getAttribute('href');
  alert(href);
  return false;
};
```

✔ Zobacz

Delegowanie. Włożone elementy

- ✔ Wewnątrz a może być inny elemet

```
<a href='/php'><strong>PHP</strong> 5.0</a>
```

- ✔ wówczas przykład nie będzie działać
- ✔ rozwiązanie:

```
document.getElementById('menu').onclick
        = function(event) {
  var target = event.target;
  while (target != this) {
    if (target.tagName == 'A') {
      alert(target.getAttribute('href'));
      return;
    }
    target = target.parentNode;
} }
```

Przykład z menu. HTML


```
<div id="menu">
  <button data-action="save">Save</button>
  <button data-action="load">Load</button>
  <button data-action="search">Search</button>
</div>
```

Przykład z menu. Javascript


```
function Menu(elem) {
  this.save = function() {alert( 'save' );};
  this.load = function() {alert( 'load' );};
  this.search = function() {alert( 'find' );};
  var self = this;
  elem.onclick = function(e) {
    var target = e.target;
    var action = target.getAttribute('data-action');
    if (action) {
      self[action]();
    }
  };
}
new Menu(menu);
```

✔ Zobacz

Wzorzec czynnościowy

Wzorzec projektowy czynnościowy

- ✔ Elementowi wystawiamy atrybut, który opisuje go zachowanie
- ✔ Na dokumencie rejestrujemy callback na kliknięcia, który odpala funkcję, jeżeli element ma odpowiedni atrybut

Przykład: licznik


```
Licznik 1:
<button data-counter>1</button>
Licznik 2:
<button data-counter>2</button>
<script>
document.onclick = function(event) {
  if (!event.target.
    hasAttribute('data-counter')) return;
  var counter = event.target;
  counter.innerHTML++;
};
</script>
```

✔ Zobacz

Przykład: pokazywanie formularza


```
<button data-toggle-id="subscribe-mail">
    Show subscribtion form </button>
<form id="subscribe-mail" hidden>
    Email: <input type="email"> </form>
<script>
  document.onclick = function(event) {
    var target = event.target;
    var id = target.getAttribute('data-toggle-id');
    if (!id) return;
    var elem = document.getElementById(id);
    elem.hidden = !elem.hidden;
  };
</script>
```

✔ Zobacz

Domyslne akcje

Powsztrzymane akcji domyślnej

- ✔ Przejście po kliknięciu na odsyłacz, pokazywanie dymka (atrybut `title`), etc
- ✔ Jeżeli domyślna akcja ma miejsce po zdarzeniu, można ją powstrzymać na dwa sposoby:

1. `event.preventDefault()`
2.

```
element.onclick = function(event) {
  ...
  return false;
}
```

   ✘ nie działa dla `addEventListener`

- ✔ przykłady: sposób 1, przez `return false`

Kilka uwag

- ✔ Powstrzymanie akcji domyślnej nie powstrzymuje propagacji
- ✔ Przy rejestracji kilku funkcji, kolejność ich przy zajściu zdarzenia nie jest określona
- ✔ Jeżeli propagacja i akcja domyślna zostały powstrzymane, na bieżącym elemencie mogą zadziałać inne funkcje reagujące
- ✔ Jeżeli domyślna akcja odbywa się przed zdarzeniem, to jej nie można powstrzymać

✘

```
onfocus="return false;"
```

nie działa

Przykład

✔ Czy po kliknięciu przeglądarka przejdzie na W3C?

```
<script>
  function handler() {
    alert("...");
    return false;
  }
</script>
<a href="http://w3.org" onclick="handler()">
  w3.org
</a>
```

Sprawdź

Zdarzenia użytkownika

Generowanie zdarzeń

- ✔ Obiekt zdarzenia generuje się konstruktorem `Event`

```
var event = new Event(typ zdarzenia[, flagi]);
```

  - ✘ typ może być stanadardowym (`click`) bądź niestandardowym
  - ✘ flagi:`{ bubbles: true/false, cancelable: true/false }`
    - ✓ domyślnie `{bubbles: false, cancelable: false}`

Odpalanie zdarzeń

- ✔ Zdarzenie można odpalić funkcją `dispatchEvent(event)`

```
<button id="elem"
    onclick="alert('Click');">Autoclick</button>

<script>
  var event = new Event("click");
  elem.dispatchEvent(event);
</script>
```

- ✔ Zobacz

Ładowanie dokumentu

Ładowanie zewnętrznych zasobów

- ✔ Przeglądarka śledzi ładowanie zewnętrznych zasobów
- ✔ Dwa zdarzenia:
  - ✘ `load`
  - ✘ `error`

Ładowanie skryptów

- ✔ utworzyć element `script`
- ✔ dodać do DOM
- ✔ po załadowaniu i wykonaniu skryptu zdarzenie `load`
- ✔ w razie problemów z załadowaniem (nie z wykonaniem) zdarzenie `error`

Ładowanie skryptów. Przykład


```
var script = document.createElement('script');
script.src =
"http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.js
document.documentElement.appendChild(script);
script.onload = function() {
  alert(jQuery);
}
script.onerror = function() {
  alert("Error: " + this.src);
}
```

- ✔ Powodzenie
- ✔ Niepowodzenie

Zdarzenia okna. error

- ✔ Funkcja `window.onerror` odpala się na każdy błąd oprócz bloków `try--throw--catch`
- ✔ Argumenty tej funkcji
  - ✘ `message`
  - ✘ `source`
  - ✘ `lineno`
- ✔ Przykład

```
window.onerror=function(message, source, lineno)
{  alert("Error: "+message +"\n" +
    "file: " + source + "\n" +
    "line: " + lineno);
};
```

Zdarzenia okna. `load`, `unload`

- ✔ Funkcja `window.onload` odpala się po załadowaniu całej strony (łacznie ze stylami, obrazkami, iframe'ami, etc)
- ✔ Funkcja `window.onunload` odpala się po zamknięciu przeglądarki, przejściu na inną stronę, etc
  - ✘ można zamknąć pop-up okna
  - ✘ nie można zatrzymać przejścia

Zdarzenia okna. beforeunload

- ✔ Funkcja `window.onbeforeunload` pozwala przejście po potwierdzeniu
  - ✘ powinna zwrócić tekst
  - ✘ Firefox ten tekst ignoruje
  - ✘ Opera nie wspiera tego zdarzenia
  - ✘ Przykład

```
window.onbeforeunload = function() {
  return "Are you sure?";
};
```

Zdarzenie DOMContentLoaded

- ✔ zdarzenie `DOMContentLoaded` następuje przy załadowaniu dokumentu, po wykonaniu wszystkich znaczników `script`
- ✔ w odróżnieniu od `load`, ono nie czeka na załadowanie dodatkowych zasobów, więc następuje znacznie wcześniej.
- ✔ w momencie wystąpienie `DOMContentLoaded`, drzewo DOM jest całkowicie załadowane i wszystkie elementy są dostępne.
- ✔ Zobaczyć

DOMContentLoaded a skrypty

- ✔ przeglądarka czeka na wykonanie wszytkich skryptów ze znaczników `script`
  - ✘ nie czeka, jeżeli jest atrybut `async/defer`
    - ✓ Opera czeka na wszystkie skrypty
  - ✘ nie czeka, jeżeli skrypt został dodany przez DOM

Drag'N'Drop

Drag'n'Drop

- ✔ za pomocą zdarzenia `mousedown` wykryć naciśnięcie myszy na elemencie
- ✔ przy naciśnięciu — przygotować obiekt:
  - ✘ dać mu `position:absolute`
  - ✘ pokazać w tym samym miejscu, ustawiając `left/top` ze współrzędnych kursora
- ✔ Śledzić przesunięcia myszki przez `mousemove` i przesuwać obiekt:
  - ✘ zmieniając `left/top`
- ✔ przy `mouseup` zatrzymać przeniesienie i zrealizować działania, związane z zakończeniem Drag'n'Drop.
- ✔ Zobaczyć

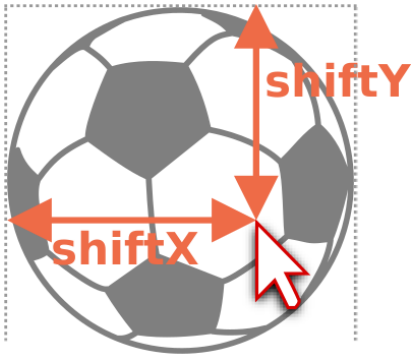
Drag'n'Drop przeglądarki

- ✔ żeby zatrzymać domyślny Drag'n'Drop przeglądarki

```
ball.ondragstart = function() {
  return false;
}
```

- ✔ Zobaczyć

Drag'n'Drop: poprawka pozycji kursora




```
shiftX = event.clientX
         - ball.getBoundingClientRect().left;
shiftY = shiftY = event.clientY
         - ball.getBoundingClientRect().top;
function moveAt(pageX, pageY) {
   ball.style.left = pageX - shiftX + 'px';
   ball.style.top = pageY - shiftY + 'px';
}
```

✔ Zobaczyć