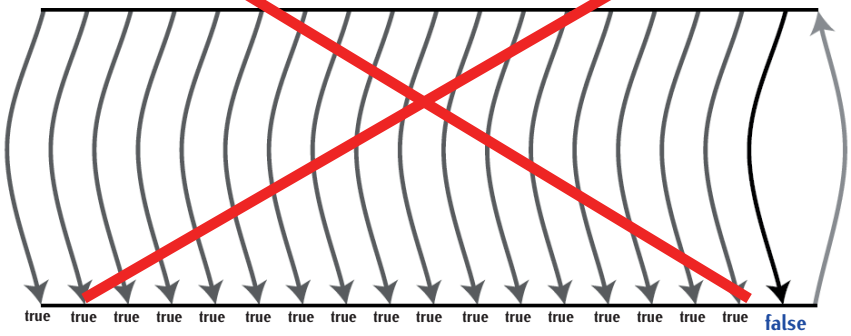
Język JavaScript

„DOM – obsługa zdarzeń”

JavaScript – brak obsługi wątków

- JavaScript **nie działa** wielowątkowo

```
” while ( ! window.loaded() ) { } // czekamy
  obsługa_zdarzenia; ”
```



Asynchroniczna obsługa zdarzeń

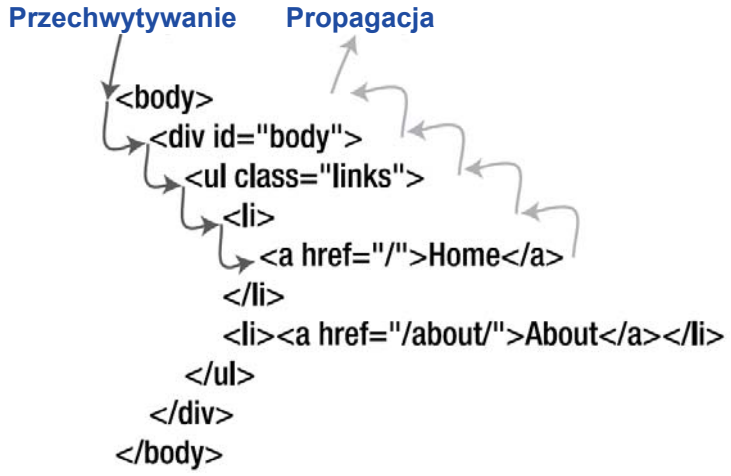
- Schemat „obsługi” zdarzenia
  - przygotowanie kodu „procedury obsługi”
  - „rejestracja” procedury obsługi zdarzenia

```
/*jshint browser: true */

// definiujemy „procedurę obsługi” zdarzenia
var loaded = function () {
    document.getElementById('body').style.border = '1px solid';
};

// Rejestrujemy procedurę przypisując ją do zdarzenia
// załadowania strony do okna przeglądarki
window.onload = loaded;
```

- Ilekroć dane zdarzenie zajdzie wykonana zostanie procedura jego obsługi

Asynchroniczna obsługa zdarzeń

- Każde zdarzenie ma dwie fazy



Prosty przykład

```
domReady(function () {
    var li = document.getElementsByTagName("li");
    Array.prototype.forEach.call(li, function (el) {
        el.onmouseover = function () {
            this.style.backgroundColor = 'blue';
        };
        el.onmouseout = function () {
            this.style.backgroundColor = 'white';
        };
    });
});
```

- Przykład ciągu zdarzeń dla **<li>** zawierającego **<a>**:
  - **<li>** mouseover: najeżdżamy myszą na **<li>**
  - **<li>** mouseout: mysz wędruje nad <a> „opuszczając” **<li>**
  - **<a>** mouseover: mysz najeżdża na **<a>**
  - **<li>** mouseover: **propagacja** zdarzenia z **<a>** na **<li>**

zmiana koloru tła **<li>** na niebieski

- **Uwaga**: metoda obsługi zdarzeń przez atrybutu „on<zdarzenie>” obsługuje jedynie fazę propagacji

Prosty przykład poprawiony

```
/*jshint browser: true */

/*global domReady: false */

domReady(function () {
    var li = document.getElementsByTagName('li'),
        toBlue = function () {
            this.style.backgroundColor = 'blue';
        },
        toWhite = function () {
            this.style.backgroundColor = 'white';
        };

    Array.prototype.forEach.call(li, function (el) {
        el.onmouseover = toBlue;
        el.onmouseout = toWhite;
    });
});
```


Ważniejsze klasy zdarzeń w DOM

- zdarzenia „myszowe”
  - **mouseover**, **mouseout**, **mouseup**, **click**, ...
- zdarzenia „klawiaturowe”
  - **keyup**, **keydown**, **keypress**
- zdarzenie „interfejsowe”
  - **focus**, **blur**
- zdarzenia „formularzowe”
  - **submit**, **change**, **select**, **reset**
- zdarzenia ładowania dokumentu
  - **load**, **unload**, **beforeunload**, ...

JavaScript – „infrastruktura” zdarzeń

- **Obiekt zdarzenia**
  - zawiera informacje na temat „kontekstu” zdarzenia
    - np. w przypadku zdarzenia keypress, możemy się z niego dowiedzieć o tym, który klawisz został naciśnięty
  - niestety istnieją dwie implementacje
    - IE: pojedynczy globalny obiekt **window.event**
    - „pozostałe przeglądarki”: zgodnie ze standardem W3C, obiekt zdarzenia przekazywany jest do procedur obsługi zdarzeń jako argument

JavaScript – „infrastruktura" zdarzeń

- Jak bezpiecznie poradzić sobie z różnicami?
- Przykład
  - chcemy, aby naciskanie klawisza Enter w obrębie pierwszego elementu **<textarea>** na stronie nie powodowało wprowadzania znaku zmiany wiersza

```
// Znajduje pierwszy element <textarea> na stronie
// i dołącza do niego obsługę zdarzenia „keypress"

document.getElementsByTagName('textarea')[0].onkeypress = function (e) {
    // jeśli obiekt zdarzenia „e" ma wartość nieokreśloną
    // (czyli działamy z IE) wykorzystujemy obiekt globalny
    e = e || window.event;

    // jeśli wciśnięto klawisz Enter zwracamy „false", co powoduje,
    // że zdarzenie to jest pomijane
    return e.keyCode !== 13;
};
```


JavaScript – „infrastruktura" zdarzeń

- Procedury obsługi – referencja **this**
  - widzieliśmy już poprzednio, że **this** odnosi się do „bieżącego elementu" zdarzenia
  - pozwala to tworzyć „generyczne" procedury obsługi

```
var handleClick = function () {
    this.style.backgroundColor = 'blue';
    this.style.color = 'white';
};

…
<p onclick='handleClick();'>
    Jakiś tekst
</p>
…
```

Błąd: **this**.style is undefined!!!

JavaScript – „infrastruktura" zdarzeń

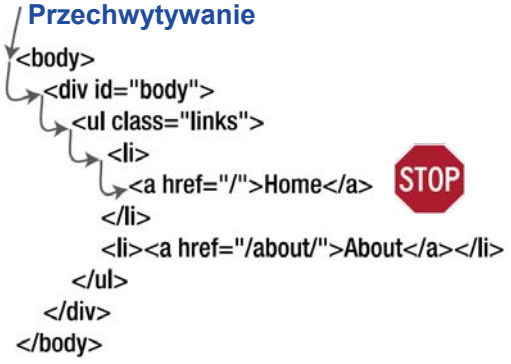
- Procedury obsługi – referencja **this**
  - musimy jawnie przekazać **this** jako argument

```
var handleClick = function (elem) {
    elem.style.backgroundColor = 'blue';
    elem.style.color = 'white';
};

…
<p onclick='handleClick(this);'>
    Jakiś tekst
</p>
…
```

Wniosek: „atrybuty zdarzeniowe" HTML-a są „**beee**" :)

JavaScript – „infrastruktura" zdarzeń

- Powstrzymywanie propagacji zdarzenia
  - jeśli chcemy, aby zdarzenie zostało obsłużone wyłącznie „w miejscu swego wystąpienia" musimy uniemożliwić jego propagację do elementów „nadrzędnych"



JavaScript – „infrastruktura" zdarzeń

- Powstrzymywanie propagacji zdarzenia
  - niestety w IE powstrzymywanie uzyskujemy inaczej niż w „pozostałych przeglądarkach"

```
var doNotPropagate = function (e) {
    // jeśli argument „e" jest zdefiniowany i oferuje metodę
    // stopPropagation to mamy do czynienia z jedną
    // z „pozostałych przeglądarek"
    if (e && e.stopPropagation) {
        // i stosujemy standard W3C
        e.stopPropagation();
    } else {
        // Działamy z IE i musimy użyc jego „standardu"
        // odwołując się do globalnego obiektu window.event
        window.event.cancelBuble = true;
    }
};
```


Powstrzymywanie propagacji – przykład

```
/*jshint browser: true */
/*global doNotPropagate: false */

var // Wyszukujemy wszystkie elementy dokumentu
    all = document.getElementsByTagName('*'),
    addHighlight = function (e) {
        this.style.backgroundColor = '#eee';
        doNotPropagate(e);
    },
    remHighlight = function (e) {
        this.style.backgroundColor = '#fff';
        doNotPropagate(e);
    };

// Dodajemy do każdego elementu procedury obsługi
// zdarzeń „mouseover" i „mouseout"
Array.prototype.forEach.call(all, function (el) {
    el.onmouseover = addHighlight;
    el.onmouseout  = remHighlight;
});
```


Domyślne akcje i ich przechwytywanie

- Domyślne akcje przeglądarek – przykłady
  - otwarcie witryny wskazywanej przez atrybut **`href`** elementu **`<a>`** po kliknięciu na odsyłacz
  - zapisanie reprezentacji strony na dysku po naciśnięciu kombinacji klawiszy (Ctrl-S)
  - wyświetlenie „chmurki" z wartością atrybut **`title`** lub **`alt`** (w zależności od przeglądarki) elementu **`<img>`** po najechaniu kursorem myszy
  - wysłanie formularza po naciśnięciu przycisku „wyślij"
  - ...

Domyślne akcje i ich „przechwytywanie"

- I znów – istnieją dwie metody przechwytywania akcji domyślnych – metoda IE oraz metoda „pozostałych przeglądarek" (zgodna ze standardem W3C)

```
var stopDefault = function (e) {
    // Zgodne ze standardem W3C powstrzymywanie
    // domyślnych akcji w „pozostałych przeglądarkach"
    if (e && e.preventDefault) {
        e.preventDefault();
    } else {
        // Działamy z IE i musimy użyć jego „standardu"
        // odwołując się do globalnego „window.event"
        window.event.returnValue = false;
    }
};
```

- funkcja **`stopDefault`** powinna być używana jako ostatni element procedury obsługi zdarzenia

Prosty przykład

```
/*jshint browser: true */
/*global domReady, stopDefault, alert */

// Wykorzystanie przechwytywania domyślnych akcji
// do „cenzurowania odsyłaczy”
domReady(function () {
    var elems = document.getElementsByTagName("a"),
        re = /^http:.*microsoft.*/,
        noWay = function (e) {
            alert('No way!!!!');
            return stopDefault(e);
        };

    Array.prototype.forEach.call(elems, function (el) {
        if (re.test(el.href)) {
            el.onclick = noWay;
        }
    });
});
```


Wiązanie procedur obsługi ze zdarzeniami

- Metoda „tradycyjna” – procedury obsługi zdarzeń dodawane są jako atrybuty elementów DOM
  - „trzy proste przykłady”

```
01. // Znajduje element <form> i dołącza do niego procedurę obsługi
02. // zdarzenia 'submit'
03. document.getElementsByTagName("form")[0].onsubmit = function(e) {
04.     // zablokuj możliwość wysyłania formularzy
05.     return stopDefault(e);
06. };
07.
08. // Dołącza procedurę obsługi zdarzenia "keypress"
09. // do elementu <body> dokumentu
10. document.body.onkeypress = myKeyPressHandler;
11.
12. // Dołącza procedurę obsługi zdarzenia "load" do dokumentu
13. window.onload = function() {…};
```


Wiązanie procedur obsługi ze zdarzeniami

- Metoda „tradycyjna” c.d.
  - zalety:
    - najprostsza
    - **`this`** zawsze wskazuje na bieżący element
    - bezproblemowa obsługa w przeglądarkach
  - ograniczenia i wady
    - obsługuje tylko fazę propagacji zdarzeń
    - pozwala na związanie z danym zdarzeniem (w kontekście konkretnego elementu) wyłącznie jednej procedury obsługi, co może prowadzić do problemów – np. przypadkowe „przesłonięcie”

Wiązanie procedur obsługi ze zdarzeniami

- Metoda standardowa (W3C)
  - korzysta z metody **`addEventListener`** dostępnej dla wszystkich elementów DOM
  - „trzy proste przykłady” w wersji W3C

```
01. // Znajduje element <form> i dołącza do niego procedurę obsługi
02. // zdarzenia 'submit'
03. document.getElementsByTagName("form")[0].addEventListener('submit', function(e) {
04.     // zablokuj możliwość wysyłania formularzy
05.     return stopDefault(e);
06. }, false);
07.
08. // Dołącza procedurę obsługi zdarzenia "keypress"
09. // do elementu <body> dokumentu
10. document.body.addEventListener('keypress', myKeyPressHandler, false);
11.
12. // Dołącza procedurę obsługi zdarzenia "load" do dokumentu
13. window.addEventListener('load', function() {…}, false);
```

Wiązanie procedur obsługi ze zdarzeniami

- Metoda standardowa (**W3C**)
  - zalety
    - działa dla fazy propagacji, oraz fazy przechwytywania zdarzeń (trzeci argument metody `addEventListener`: `false` to *faza propagacji*, a `true` – *faza przechwytywania*)
    - wewnątrz procedur obsługi zdarzeń `this` zawsze oznacza bieżący element
    - obiekt zdarzenia jest zawsze dostępny jako pierwszy argument procedury obsługi
    - pozwala skojarzyć dowolną liczbę procedur obsługi
  - ograniczenia / wady
    - nie działa z przeglądarkami **IE < 9**

Wiązanie procedur obsługi ze zdarzeniami

- Metoda IE
  - podobna do metody standardowej, wykorzystuje metodę `attachEvent`
  - „trzy proste przykłady” w wersji IE

```
01. // Znajduje element <form> i dołącza do niego procedurę obsługi
02. // zdarzenia 'submit'
03. document.getElementsByTagName("form")[0].attachEvent('onsubmit', function() {
04.     // zablokuj możliwość wysyłania formularzy
05.     return stopDefault();
06. });
07.
08. // Dołącza procedurę obsługi zdarzenia "keypress"
09. // do elementu <body> dokumentu
10. document.body.attachEvent('onkeypress', myKeyPressHandler);
11.
12. // Dołącza procedurę obsługi zdarzenia "load" do dokumentu
13. window.attachEvent('onload', function() {…});
```


Wiązanie procedur obsługi ze zdarzeniami

- Metoda IE
  - zalety
    - z danym zdarzeniem można skojarzyć dowolną liczbę procedur obsługi
  - ograniczenia i wady
    - działa wyłącznie dla fazy propagacji zdarzeń
    - wewnątrz procedur obsługi `this` zamiast bieżącego elementu oznacza obiekt `window`
    - globalny obiekt `window.event`
    - dostępna wyłącznie w przeglądarkach **IE** oraz Opera

Obsługa zdarzeń a biblioteki JavaScript

- Obsługa zdarzeń w przeglądarkach wciąż daleka od ideału
- Bezpieczna alternatywa – biblioteki JavaScript
  - abstrahują oraz wzbogacają model zdarzeniowy (niekiedy umożliwiając definiowanie własnych zdarzeń oraz ich obsługi)
  - oferują różnego rodzaju funkcje „użytkowe”
- Omówimy krótko obsługę zdarzeń w bibliotekach
  - jQuery
  - YUI Event Utility

Obsługa zdarzeń a biblioteki JavaScript

- jQuery
  - oferuje ujednolicony sposób definiowania procedur obsługi zdarzeń
  - pozwala dołączyć dowolną liczbę procedur obsługi do danego zdarzenia dla każdego elementu DOM
  - udostępnia „znormalizowany” obiekt zdarzenia jako argument procedurom obsługi
  - oferuje zunifikowane metody „kasowania” zdarzeń oraz przechwytywania/blokowania akcji domyślnych

Obsługa zdarzeń a biblioteki JavaScript

- jQuery – kilka przykładów

```
01. // wykonuje funkcję dopiero, gdy drzewo DOM jest już utworzone
02. $(document).ready(function() {...});
03.
04. // dołącza prostą procedurę obsługi zdarzenia "click" do
05. // wszystkich elementów <img> na stronie
06. $('img').bind('click',function(event){alert('Hi there!');});
07.
08. // automatycznie zmienia klasę wiersza tabeli zawierającej dany
09. // element na "Over" w momencie, gdy nad elementem znajduje się
10. // kursor myszy
11. $(this).closest('tr').bind("mouseenter mouseleave", function(e) {
12.     $(this).toggleClass("Over");
13. });
14.
15. // wyświetla informacje o współrzędnych miejsca kliknięcia elementu
16. // oraz nazwie elementu na którym kliknięto dwukrotnie
17. $("p").bind("click",
18. function(e) {
19.     var str = "( " + e.pageX + ", " + e.pageY + " )";
20.     $("span").text("Click happened! " + str);
21. });
22. $("p").bind("dblclick",
23. function() {
24.     $("span").text("Double-click happened in " + this.tagName);
25. });
```


Obsługa zdarzeń a biblioteki JavaScript

- YUI Event Utility
  - oferuje elastyczne mechanizmy dołączania i usuwania procedur obsługi zdarzeń do/z elementów DOM
  - pozwala automatycznie opóźnić dołączenie procedur obsługi do chwili, gdy dany element DOM jest już dostępny
  - udostępnia obiekt zdarzenia jako argument procedurom obsługi oraz umożliwia dołączenie do niego dowolnego obiektu
  - „normalizuje” obiekt zdarzenia udostępniając najczęściej używane właściwości (za pomocą odpowiednich metod)
  - udostępnia mechanizm definiowania i tworzenia obsługi własnych zdarzeń

Obsługa zdarzeń a biblioteki JavaScript

- YUI Event Utility – kilka przykładów

```
01. // dołączanie procedury obsługi do elementu
02. var oElement = document.getElementById("elementid");
03. function fnCallback(e) { alert("click"); }
04. YAHOO.util.Event.addListener(oElement, "click", fnCallback);
05.
06. // dołączanie procedury obsługi do grupy elementów
07. var ids = ["el1", "el2", "el3"];
08. function fnCallback(e) { alert(this.id); }
09. YAHOO.util.Event.addListener(ids, "click", fnCallback);
10.
11. // "odpytywanie" elementu o związane z nim procedury obsługi zdarzeń
12. // (wszystkich typów)
13. var listeners = YAHOO.util.Event.getListeners(myelement);
14. for (var i=0; i<listeners.length; ++i) {
15.     var listener = listeners[i];
16.     alert( listener.type   ); // typ zdarzenia
17.     alert( listener.fn     ); // procedura obsługi
18. }
19.
20. // tylko procedury obsługi zdarzenia "click"
21. var listeners = YAHOO.util.Event.getListeners(myelement, "click");
```

CSS i JavaScript działają inaczej!

- Stwierdzenie niemal całkowicie trywialne...
- CSS działa „zawsze”
  - reguły stylu CSS aplikowane są do każdego elementu drzewa DOM, bez względu na to kiedy element ten został do niego dodany
- JavaScript pozwala modyfikować styl elementów tylko w reakcji na zajście zdarzenia
  - jeśli chcemy „związać” procedurę obsługi również „z nowo dodawanymi” elementami możemy zastosować zasadę delegowania – zdarzenie jest obsługiwane na poziomie elementu otaczającego

Delegowanie obsługi zdarzeń

Krótkie „demo”