

jQuery

v3.0.0

Plan

- Wprowadzenie
- Szukanie elementów w jQuery
- Eventy
- Manipulacja DOM-em
- Eventy zaawansowane

Wprowadzenie

Czym jest jQuery?

jQuery jest **biblioteką** napisaną w języku JavaScript służącą do operacji na drzewie DOM i ujednoliceniu działania stron na różnych przeglądarkach.

Najważniejsze funkcje, które implementuje jQuery:

- łatwiejsze wyszukiwanie elementów,
- lepsza kontrola dodawania i usuwania elementów,
- animacje,
- łatwiejsze w użyciu eventy,
- Ajax.

JavaScript a jQuery

JavaScript

```
var x, y;
if (self.innerHeight) { /* all except Explorer */
    x = self.innerWidth;
    y = self.innerHeight;
}
else if (document.documentElement &&
    document.documentElement.clientHeight) {
    /* Explorer 6 Strict Mode */
    x = document.documentElement.clientWidth;
    y = document.documentElement.clientHeight;
}
else if (document.body) { /* other explorers */
    x = document.body.clientWidth;
    y = document.body.clientHeight;
}
```

JavaScript a jQuery

jQuery

```
var x = $(window).width();  
var y = $(window).height();
```

JavaScript a jQuery

jQuery

```
var x = $(window).width();  
var y = $(window).height();
```

Oba przykłady kodu pokazują sposób na pobranie szerokości i wysokości okna.

Instalacja jQuery

- Ściągnij pliki ze strony <http://jquery.com/download>
- Dodaj ściągnięty skrypt do kodu HTML za pomocą tagu **script**, kolejność jest ważna – **jQuery** powinno być **przed** naszymi skryptami,
- Można też korzystać z plików hostowanych przez **np. Google (tak zwane CDN)**
<https://cdnjs.com/libraries/jquery>

Dokumentacja

- jQuery ma bardzo dobrze napisaną dokumentację. Można tam znaleźć dokładne opisy, przykłady użycia i najczęstsze problemy z danymi metodami.
- Przed użyciem każdej metody należy chociaż pobieżnie przeczytać jej dokumentację.
- <http://api.jquery.com>

Wersje jQuery

Aktualnie dostępna w trzech wersjach:

- 1.x – wspierająca przeglądarki IE6/7/8.
- 2.x – wspierająca tylko nowsze przeglądarki.
- 3.x – wspierająca tylko nowsze przeglądarki, posiadająca nowe funkcje

⋮ względem wersji 2.

Szukanie elementów w jQuery

DOMContentLoaded w jQuery

- Zanim przystąpimy do pisania skryptów korzystających z elementów DOM, musimy sprawdzić, czy został on załadowany.
- W **Vanilla JS (czysty JavaScript)** korzystamy m.in. z **DOMContentLoaded**.
- W jQuery możemy wykorzystać następujące dwa sposoby przedstawione po prawej stronie.

```
$(function() {  
    /* tu nasz kod */  
});  
/* lub */  
$(document).ready(function() {  
    /* tu nasz kod */  
});
```

DOMContentLoaded w jQuery

- Zanim przystąpimy do pisania skryptów korzystających z elementów DOM, musimy sprawdzić, czy został on załadowany.
- W **Vanilla JS (czysty JavaScript)** korzystamy m.in. z **DOMContentLoaded**.
- W jQuery możemy wykorzystać następujące dwa sposoby przedstawione po prawej stronie.

```
$(function() {  
    /* tu nasz kod */  
});  
/* lub */  
$(document).ready(function() {  
    /* tu nasz kod */  
});
```

Wszystkie zadania i przykłady będziemy pisać wewnątrz jednej z takich funkcji.

DOMContentLoaded w jQuery

- Zanim przystąpimy do pisania skryptów korzystających z elementów DOM, musimy sprawdzić, czy został on załadowany.
- W **Vanilla JS (czysty JavaScript)** korzystamy m.in. z **DOMContentLoaded**.
- W jQuery możemy wykorzystać następujące dwa sposoby przedstawione po prawej stronie.

```
$(function() {  
    /* tu nasz kod */  
});  
/* lub */  
$(document).ready(function() {  
    /* tu nasz kod */  
});
```

Wszystkie zadania i przykłady będziemy pisać wewnątrz jednej z takich funkcji.

Wyszukiwanie elementów

- Kiedy korzystamy z jQuery, mamy możliwość łatwego wyszukiwania elementów za pomocą zapytań CSS.

Aby swobodnie korzystać z jQuery trzeba dobrze opanować wyszukiwanie elementów.

```
$( '#top' );  
$( 'li' );  
$( 'ul li' );  
$( '.boxes' );
```

Wyszukiwanie elementów

- Kiedy korzystamy z jQuery, mamy możliwość łatwego wyszukiwania elementów za pomocą zapytań CSS.

Aby swobodnie korzystać z jQuery trzeba dobrze opanować wyszukiwanie elementów.

```
$('#top');  
$('li');  
$('ul li');  
$('.boxes');
```

Znajdź element o **id top**

Wyszukiwanie elementów

- Kiedy korzystamy z jQuery, mamy możliwość łatwego wyszukiwania elementów za pomocą zapytań CSS.

Aby swobodnie korzystać z jQuery trzeba dobrze opanować wyszukiwanie elementów.

```
$( '#top' );  
$( 'li' );  
$( 'ul li' );  
$( '.boxes' );
```

Znajdź **wszystkie** elementy **li**

Wyszukiwanie elementów

- Kiedy korzystamy z jQuery, mamy możliwość łatwego wyszukiwania elementów za pomocą zapytań CSS.

Aby swobodnie korzystać z jQuery trzeba dobrze opanować wyszukiwanie elementów.

```
$( '#top' );  
$( 'li' );  
$( 'ul li' );  
$( '.boxes' );
```

Znajdź **wszystkie** elementy **li** będące potomkami elementu **ul**

Wyszukiwanie elementów

- Kiedy korzystamy z jQuery, mamy możliwość łatwego wyszukiwania elementów za pomocą zapytań CSS.

Aby swobodnie korzystać z jQuery trzeba dobrze opanować wyszukiwanie elementów.

```
$( '#top' );  
$( 'li' );  
$( 'ul li' );  
$( '.boxes' );
```

Znajdź **wszystkie** elementy z klasą **boxes**

Wyszukiwanie elementów

- Kiedy korzystamy z jQuery, mamy możliwość łatwego wyszukiwania elementów za pomocą zapytań CSS.

Aby swobodnie korzystać z jQuery trzeba dobrze opanować wyszukiwanie elementów.

```
$( '#top' );  
$( 'li' );  
$( 'ul li' );  
$( '.boxes' );
```

jQuery nie ogranicza nas w żaden sposób i możemy wykorzystać dowolnie skomplikowane selektory css.

Wyszukiwanie elementów

- W jakim celu opakowujemy selektory CSS funkcją jQuery?
- Wyszukiwania elementów za pomocą selektorów CSS daje nam ogromne możliwości, których nie ma w czystym JavaScriptcie.

Ile elementów zostanie znalezionych?

```
$ ( '#top' );
```

Wyszukiwanie elementów

- W jakim celu opakowujemy selektory CSS funkcją jQuery?
- Wyszukiwania elementów za pomocą selektorów CSS daje nam ogromne możliwości, których nie ma w czystym JavaScriptcie.

Ile elementów zostanie znalezionych?

```
$('#top');
```

Znaleziony będzie 1 element (ponieważ id), ale jQuery zawsze wyszukuje tyle elementów ile jest w stanie dopasować do selektora.

jQuery zwraca kolekcję elementów, nie jest to tablica!

Wyszukiwanie elementów

- W podanym przykładzie jQuery napotyka na selektor **id**, uruchamia zatem funkcję **JavaScript**:

```
document.getElementById("top")
```

- Jeśli selektor jest bardziej zaawansowany, jQuery uruchamia silnik **Sizzle** służący do wyszukiwania elementów DOM.

```
$("#top");
```

Wyszukiwanie elementów

- W podanym przykładzie jQuery napotyka na selektor **id**, uruchamia zatem funkcję **JavaScript**:

```
document.getElementById("top")
```

- Jeśli selektor jest bardziej zaawansowany, jQuery uruchamia silnik **Sizzle** służący do wyszukiwania elementów DOM.

```
$("#top");
```

```
#top = document.getElementById("top")
```

Wyszukiwanie elementów

- jQuery uruchamia standardowe funkcje JavaScript, jeśli przeglądarka je obsługuje.
- W poniższym przypadku uruchomiona zostanie funkcja **querySelectorAll()**
- W przypadku starszych przeglądarek np. IE8, która nie obsługuje funkcji **querySelectorAll()**, jQuery skorzysta z silnika **Sizzle**.

- Ważne aby przypisywać znalezione elementy do zmiennych, jeśli chcemy ich potem wielokrotnie użyć, zapobiegnie to ponownemu przeszukaniu drzewa DOM. Jest to tzw. **caching**.

```
var linksOnTop = $('#top a');
```


Wyszukiwanie elementów

- Musimy pamiętać, że jQuery zawsze zwraca tyle elementów ile jest w stanie dopasować do selektora
- jQuery zwraca kolekcję elementów, nie jest to tablica!

```
var linksOnTop = $('.someList li');  
//linksOnTop to kolekcja obiektów jQuery
```

Wyszukiwanie elementów

Czy możemy w takim razie odwołać się do kolekcji jak do tablicy?

```
var linksOnTop = $('.someList li');  
var thirdLi = linksOnTop[2]; // obiekt JavaScript NIE jQuery  
thirdLi.classList.add('exampleClass');  
// możemy wykonać na nim dowolne operacje JS
```

Wyszukiwanie elementów

Czy możemy w takim razie odwołać się do kolekcji jak do tablicy?

```
var linksOnTop = $('.someList li');  
var thirdLi = linksOnTop[2]; // obiekt JavaScript NIE jQuery  
thirdLi.classList.add('exampleClass');  
// możemy wykonać na nim dowolne operacje JS
```

Odwołanie się do kolekcji jQuery jak do tablicy, zwraca element JavaScript o danym indeksie, nie oznacza to, że na kolekcji można wykonać metody tablicowe!

Wyszukiwanie elementów

Jak pobrać konkretny element kolekcji?

```
var linksOnTop = $('.someList li');  
var thirdLi = linksOnTop.eq(2); // obiekt jQuery  
thirdLi.hide();  
//możemy wykonać na nim dowolne operacje jQuery, tutaj ukryć
```

Wyszukiwanie elementów

Jak pobrać konkretny element kolekcji?

```
var linksOnTop = $('.someList li');  
var thirdLi = linksOnTop.eq(2); // obiekt jQuery  
thirdLi.hide();  
//możemy wykonać na nim dowolne operacje jQuery, tutaj ukryć
```

Aby w jQuery pobrać element kolekcji używamy metody **eq()** podając jako argument indeks elementu, który chcemy pobrać

Optymalizacja wyszukiwania

Jak jQuery interpretuje ten selektor?

- jQuery przeszukuje od prawej do lewej, chyba że pierwszy element to ID.
- W tym przykładzie jQuery znajduje wszystkie elementy **a**, następnie sprawdza, które z nich mają rodzica z klasą menu.
- Nieoptymalnie, prawda?

```
$( '.menu a' );
```

Optymalizacja wyszukiwania

- A co, gdybyśmy mieli na stronie 100 linków i tylko pięć z nich znajdowałoby się w menu?
- jQuery i tak przeszuka cały dokument.
- Można zoptymalizować wyszukiwanie na kilka sposobów.

```
$( '.menu a' );
```

```
$( '.menu' ).find( 'a' );  
$( 'a', $( '.menu' ) );  
$( '#menu a' );
```

Optymalizacja wyszukiwania

- A co, gdybyśmy mieli na stronie 100 linków i tylko pięć z nich znajdowałoby się w menu?
- jQuery i tak przeszuka cały dokument.
- Można zoptymalizować wyszukiwanie na kilka sposobów.

```
$( '.menu a' );
```

```
$( '.menu' ).find( 'a' );  
$( 'a', $( '.menu' ) );  
$( '#menu a' );
```

Użyć funkcji np. **find()** lub **children()**

Optymalizacja wyszukiwania

- A co, gdybyśmy mieli na stronie 100 linków i tylko pięć z nich znajdowałoby się w menu?
- jQuery i tak przeszuka cały dokument.
- Można zoptymalizować wyszukiwanie na kilka sposobów.

```
$( '.menu a' );
```

```
$( '.menu' ).find( 'a' );  
$( 'a', $( '.menu' ) );  
$( '#menu a' );
```

Zagnieździć selektor

Optymalizacja wyszukiwania

- A co, gdybyśmy mieli na stronie 100 linków i tylko pięć z nich znajdowałoby się w menu?
- jQuery i tak przeszuka cały dokument.
- Można zoptymalizować wyszukiwanie na kilka sposobów.

```
$( '.menu a' );
```

```
$( '.menu' ).find( 'a' );  
$( 'a', $( '.menu' ) );  
$( '#menu a' );
```

Zmienić klasę na id - tutaj wymaga ingerencji w kod html

Co możemy zrobić z tymi elementami?

Po co wyszukujemy elementy?
Dlaczego jest to ważne?

Wyszukujemy elementy aby:

- nimi manipulować,
- animować je,
- usuwać,
- zmieniać,
- nadawać style lub klasy,
- itp.

```
$( '.menu' ).find( 'a' ).css( 'color', 'red' );  
$( '.menu' ).find( 'a' ).addClass( 'crazyColors' );  
$( '.menu' ).find( 'a' ).removeClass( 'crazyColors' );  
$( '.menu' ).find( 'a' ).toggleClass( 'crazyColors' );
```

Co możemy zrobić z tymi elementami?

Po co wyszukujemy elementy?
Dlaczego jest to ważne?

Wyszukujemy elementy aby:

- nimi manipulować,
- animować je,
- usuwać,
- zmieniać,
- nadawać style lub klasy,
- itp.

```
$('.menu').find('a').css('color', 'red');  
$('.menu').find('a').addClass('crazyColors');  
$('.menu').find('a').removeClass('crazyColors');  
$('.menu').find('a').toggleClass('crazyColors');
```

Metoda `css()`, dodaje/nadpisuje podaną własność css o podanej wartości

Co możemy zrobić z tymi elementami?

Po co wyszukujemy elementy?
Dlaczego jest to ważne?

Wyszukujemy elementy aby:

- nimi manipulować,
- animować je,
- usuwać,
- zmieniać,
- nadawać style lub klasy,
- itp.

```
$( '.menu' ).find( 'a' ).css( 'color', 'red' );  
$( '.menu' ).find( 'a' ).addClass( 'crazyColors' );  
$( '.menu' ).find( 'a' ).removeClass( 'crazyColors' );  
$( '.menu' ).find( 'a' ).toggleClass( 'crazyColors' );
```

Dodaje klasę do elementu/ów

Co możemy zrobić z tymi elementami?

Po co wyszukujemy elementy?
Dlaczego jest to ważne?

Wyszukujemy elementy aby:

- nimi manipulować,
- animować je,
- usuwać,
- zmieniać,
- nadawać style lub klasy,
- itp.

```
$( '.menu' ).find( 'a' ).css( 'color', 'red' );  
$( '.menu' ).find( 'a' ).addClass( 'crazyColors' );  
$( '.menu' ).find( 'a' ).removeClass( 'crazyColors' );  
$( '.menu' ).find( 'a' ).toggleClass( 'crazyColors' );
```

Usuwa klasę z elementu/ów

Co możemy zrobić z tymi elementami?

Po co wyszukujemy elementy?
Dlaczego jest to ważne?

Wyszukujemy elementy aby:

- nimi manipulować,
- animować je,
- usuwać,
- zmieniać,
- nadawać style lub klasy,
- itp.

```
$( '.menu' ).find( 'a' ).css( 'color', 'red' );  
$( '.menu' ).find( 'a' ).addClass( 'crazyColors' );  
$( '.menu' ).find( 'a' ).removeClass( 'crazyColors' );  
$( '.menu' ).find( 'a' ).toggleClass( 'crazyColors' );
```

Przełącza klasę z elementu/ów

Co możemy zrobić z tymi elementami?

Po co wyszukujemy elementy?
Dlaczego jest to ważne?

Wyszukujemy elementy aby:

- nimi manipulować,
- animować je,
- usuwać,
- zmieniać,
- nadawać style lub klasy,
- itp.

```
$('.menu').find('a').css('color', 'red');  
$('.menu').find('a').addClass('crazyColors');  
$('.menu').find('a').removeClass('crazyColors');  
$('.menu').find('a').toggleClass('crazyColors');
```

jQuery zawsze wykonuje zmianę na wszystkich elementach, nie musimy iterować :)

Czy element ma klasę?

Możemy również sprawdzić, czy dany element ma klasę za pomocą funkcji **hasClass()**

```
if ($('.menu').hasClass('crazyColors')) {  
    console.log('Menu ma klasę crazyColors');  
} else {  
    console.log('Menu nie ma klasy crazyColors');  
}
```

Czy element ma klasę?

Możemy również sprawdzić, czy dany element ma klasę za pomocą funkcji **hasClass()**

```
if ($('#menu').hasClass('crazyColors')) {  
    console.log('Menu ma klasę crazyColors');  
} else {  
    console.log('Menu nie ma klasy crazyColors');  
}
```

Jak funkcja zachowa się jeśli będziemy mieć na stronie 10 elementów o klasie menu?

Czy element ma klasę?

Możemy również sprawdzić, czy dany element ma klasę za pomocą funkcji `hasClass()`

```
if ($('.menu').hasClass('crazyColors')) {  
    console.log('Menu ma klasę crazyColors');  
} else {  
    console.log('Menu nie ma klasy crazyColors');  
}
```

Sprawdzi czy **którykolwiek** z elementów posiada tą klasę.

Czy element może zniknąć?

Funkcje powodujące przenikanie:

- **fadeIn()** – pojawienie się ukrytego elementu z efektem przenikania
- **fadeOut()** – zniknięcie widocznego elementu z efektem przenikania

```
$( '.menu' ).find( 'a' ).fadeIn( 'slow' );  
$( '.menu' ).find( 'a' ).fadeOut( 1000 );
```

Czy element może zniknąć?

Funkcje powodujące przenikanie:

- **fadeIn()** – pojawienie się ukrytego elementu z efektem przenikania
- **fadeOut()** – zniknięcie widocznego elementu z efektem przenikania

```
$('.menu').find('a').fadeIn('slow');  
$('.menu').find('a').fadeOut(1000);
```

Jako argument funkcje z rodziny **fade** przyjmują wartości napisowe **slow** lub **fast** lub czas w milisekundach

Ustawianie atrybutów elementów?

Możemy pobierać i ustawiać atrybuty elementów np.

- **class**,
- **id**,
- **type**
- i inne, za pomocą funkcji **attr()**

```
<input value="" type="text"  
        class="user-name">
```

```
var userName = $('input.user-name');  
userName.attr('type');  
userName.attr('type', 'password');
```

Ustawianie atrybutów elementów?

Możemy pobierać i ustawiać atrybuty elementów np.

- **class**,
- **id**,
- **type**
- i inne, za pomocą funkcji **attr()**

```
<input value="" type="text"  
        class="user-name">
```

```
var userName = $('input.user-name');  
userName.attr('type');  
userName.attr('type', 'password');
```

Pobranie.

Co w sytuacji jeśli elementów będzie więcej niż 1?

Ustawianie atrybutów elementów?

Możemy pobierać i ustawiać atrybuty elementów np.

- **class**,
- **id**,
- **type**
- i inne, za pomocą funkcji **attr()**

```
<input value="" type="text"  
      class="user-name">
```

```
var userName = $('input.user-name');  
userName.attr('type');  
userName.attr('type', 'password');
```

Zwrócony zostanie atrybut **pierwszego** elementu

Ustawianie atrybutów elementów?

Możemy pobierać i ustawiać atrybuty elementów np.

- **class**,
- **id**,
- **type**
- i inne, za pomocą funkcji **attr()**

```
<input value="" type="text"  
        class="user-name">
```

```
var userName = $('input.user-name');  
userName.attr('type');  
userName.attr('type', 'password');
```

Ustawienie

```
<input value="" type="password"  
        class="user-name">
```

Ustawienie atrybutu działa na wszystkie elementy, nie tylko pierwszy.

each()

Oto jedna z ważniejszych i popularniejszych funkcji jQuery: **each(index, element)**

Jest bardzo wygodna przy rozbudowanej strukturze DOM-u. Funkcja to tak naprawdę pętla, która iteruje po elementach znalezionych w DOM-ie.

index zawiera numer kolejnego elementu, a **element** to obiekt tego elementu.

Kod HTML

```
<a href="http://paste.it">Paste</a>
<a href="http://codepen.io">CodePen</a>
<a href="http://jsbin.com">JS Bin</a>
```

Kod JavaScript

```
var links = $('a');
links.each(function (index, element) {
    console.log($(element).attr('href'))
    //lub
    console.log($(this).attr('href'));
});
```

each()

Oto jedna z ważniejszych i popularniejszych funkcji jQuery: **each(index, element)**

Jest bardzo wygodna przy rozbudowanej strukturze DOM-u. Funkcja to tak naprawdę pętla, która iteruje po elementach znalezionych w DOM-ie.

index zawiera numer kolejnego elementu, a **element** to obiekt tego elementu.

Kod HTML

```
<a href="http://paste.it">Paste</a>
<a href="http://codepen.io">CodePen</a>
<a href="http://jsbin.com">JS Bin</a>
```

Kod JavaScript

```
var links = $('a');
links.each(function (index, element) {
    console.log($(element).attr('href'))
    //lub
    console.log($(this).attr('href'));
});
```

Pobieramy wszystkie linki

each()

Oto jedna z ważniejszych i popularniejszych funkcji jQuery: **each(index, element)**

Jest bardzo wygodna przy rozbudowanej strukturze DOM-u. Funkcja to tak naprawdę pętla, która iteruje po elementach znalezionych w DOM-ie.

index zawiera numer kolejnego elementu, a **element** to obiekt tego elementu.

Kod HTML

```
<a href="http://paste.it">Paste</a>
<a href="http://codepen.io">CodePen</a>
<a href="http://jsbin.com">JS Bin</a>
```

Kod JavaScript

```
var links = $('a');
links.each(function (index, element) {
    console.log($(element).attr('href'))
    //lub
    console.log($(this).attr('href'));
});
```

Iterujemy po kolekcji linków, w zmiennej **index** znajdzie się kolejny indeks elementu a w zmiennej **element** pojedynczy element

each()

Oto jedna z ważniejszych i popularniejszych funkcji jQuery: **each(index, element)**

Jest bardzo wygodna przy rozbudowanej strukturze DOM-u. Funkcja to tak naprawdę pętla, która iteruje po elementach znalezionych w DOM-ie.

index zawiera numer kolejnego elementu, a **element** to obiekt tego elementu.

Kod HTML

```
<a href="http://paste.it">Paste</a>
<a href="http://codepen.io">CodePen</a>
<a href="http://jsbin.com">JS Bin</a>
```

Kod JavaScript

```
var links = $('a');
links.each(function (index, element) {
    console.log($(element).attr('href'))
    //lub
    console.log($(this).attr('href'));
});
```

this - w jQuery działa również **this** reprezentujące bieżący element ale musimy stworzyć z niego obiekt jQuery poprzez **\$(this)**

Zadania

Czas na zadania

Zaawansowane wyszukiwanie elementów

Metody do wyszukiwania elementów

Wszystkie te metody mogą jako argument przyjąć selektor css, który filtruje znalezione elementy

- **find()** – znajduje elementy, które są zagnieżdżone w innym
- **closest()** – znajduje elementy najbliższe idąc w górę drzewa DOM
- **children()** – znajduje wszystkie dzieci danego elementu
- **parent()** – znajduje rodzica elementu
- **siblings()** – znajduje rodzeństwo elementu
- **next()** – znajduje następny element
- **prev()** – znajduje poprzedni element

```
$( 'div' ).find();  
$( 'div' ).find( 'p.head' );  
$( 'div' ).parent();  
$( 'div' ).parent( 'li' );  
$( 'div' ).siblings();  
$( 'div' ).siblings( 'span' );  
$( 'div' ).next();  
$( 'div' ).next( 'h1' );
```


Zaawansowane wyszukiwanie elementów

Metody do wyszukiwania elementów

Wszystkie te metody mogą jako argument przyjąć selektor css, który filtruje znalezione elementy

- **find()** – znajduje elementy, które są zagnieżdżone w innym
- **closest()** – znajduje elementy najbliższe idąc w górę drzewa DOM
- **children()** – znajduje wszystkie dzieci danego elementu
- **parent()** – znajduje rodzica elementu
- **siblings()** – znajduje rodzeństwo elementu
- **next()** – znajduje następny element
- **prev()** – znajduje poprzedni element

```
$( 'div' ).find();  
$( 'div' ).find( 'p.head' );  
$( 'div' ).parent();  
$( 'div' ).parent( 'li' );  
$( 'div' ).siblings();  
$( 'div' ).siblings( 'span' );  
$( 'div' ).next();  
$( 'div' ).next( 'h1' );
```

Znajdzie wszystkie elementy we wszystkich elementach **div**

Zaawansowane wyszukiwanie elementów

Metody do wyszukiwania elementów

Wszystkie te metody mogą jako argument przyjąć selektor css, który filtruje znalezione elementy

- **find()** – znajduje elementy, które są zagnieżdżone w innym
- **closest()** – znajduje elementy najbliższe idąc w górę drzewa DOM
- **children()** – znajduje wszystkie dzieci danego elementu
- **parent()** – znajduje rodzica elementu
- **siblings()** – znajduje rodzeństwo elementu
- **next()** – znajduje następny element
- **prev()** – znajduje poprzedni element

```
$( 'div' ).find();  
$( 'div' ).find( 'p.head' );  
$( 'div' ).parent();  
$( 'div' ).parent( 'li' );  
$( 'div' ).siblings();  
$( 'div' ).siblings( 'span' );  
$( 'div' ).next();  
$( 'div' ).next( 'h1' );
```

Znajdzie wszystkie paragrafy o klasie **head** we wszystkich elementach **div**

Zaawansowane wyszukiwanie elementów

Metody do wyszukiwania elementów

Wszystkie te metody mogą jako argument przyjąć selektor css, który filtruje znalezione elementy

- **find()** – znajduje elementy, które są zagnieżdżone w innym
- **closest()** – znajduje elementy najbliższe idąc w górę drzewa DOM
- **children()** – znajduje wszystkie dzieci danego elementu
- **parent()** – znajduje rodzica elementu
- **siblings()** – znajduje rodzeństwo elementu
- **next()** – znajduje następny element
- **prev()** – znajduje poprzedni element

```
$('div').find();  
$('div').find('p.head');  
$('div').parent();  
$('div').parent('li');  
$('div').siblings();  
$('div').siblings('span');  
$('div').next();  
$('div').next('h1');
```

Znajdzie rodzica wszystkich elementów **div**

Zaawansowane wyszukiwanie elementów

Metody do wyszukiwania elementów

Wszystkie te metody mogą jako argument przyjąć selektor css, który filtruje znalezione elementy

- **find()** – znajduje elementy, które są zagnieżdżone w innym
- **closest()** – znajduje elementy najbliższe idąc w górę drzewa DOM
- **children()** – znajduje wszystkie dzieci danego elementu
- **parent()** – znajduje rodzica elementu
- **siblings()** – znajduje rodzeństwo elementu
- **next()** – znajduje następny element
- **prev()** – znajduje poprzedni element

```
$( 'div' ).find();  
$( 'div' ).find( 'p.head' );  
$( 'div' ).parent();  
$( 'div' ).parent( 'li' );  
$( 'div' ).siblings();  
$( 'div' ).siblings( 'span' );  
$( 'div' ).next();  
$( 'div' ).next( 'h1' );
```

Znajdzie rodzica wszystkich elementów **div** o ile jest nim element **li**

Zaawansowane wyszukiwanie elementów

Metody do wyszukiwania elementów

Wszystkie te metody mogą jako argument przyjąć selektor css, który filtruje znalezione elementy

- **find()** – znajduje elementy, które są zagnieżdżone w innym
- **closest()** – znajduje elementy najbliższe idąc w górę drzewa DOM
- **children()** – znajduje wszystkie dzieci danego elementu
- **parent()** – znajduje rodzica elementu
- **siblings()** – znajduje rodzeństwo elementu
- **next()** – znajduje następny element
- **prev()** – znajduje poprzedni element

```
$( 'div' ).find();  
$( 'div' ).find( 'p.head' );  
$( 'div' ).parent();  
$( 'div' ).parent( 'li' );  
$( 'div' ).siblings();  
$( 'div' ).siblings( 'span' );  
$( 'div' ).next();  
$( 'div' ).next( 'h1' );
```

Znajdzie wszystkie elementy rodzeństwa, wszystkich elementów **div**

Zaawansowane wyszukiwanie elementów

Metody do wyszukiwania elementów

Wszystkie te metody mogą jako argument przyjąć selektor css, który filtruje znalezione elementy

- **find()** – znajduje elementy, które są zagnieżdżone w innym
- **closest()** – znajduje elementy najbliższe idąc w górę drzewa DOM
- **children()** – znajduje wszystkie dzieci danego elementu
- **parent()** – znajduje rodzica elementu
- **siblings()** – znajduje rodzeństwo elementu
- **next()** – znajduje następny element
- **prev()** – znajduje poprzedni element

```
$( 'div' ).find();  
$( 'div' ).find( 'p.head' );  
$( 'div' ).parent();  
$( 'div' ).parent( 'li' );  
$( 'div' ).siblings();  
$( 'div' ).siblings( 'span' );  
$( 'div' ).next();  
$( 'div' ).next( 'h1' );
```

Znajdzie wszystkie elementy rodzeństwa będące elementami **span**, wszystkich elementów **div**

Zaawansowane wyszukiwanie elementów

Metody do wyszukiwania elementów

Wszystkie te metody mogą jako argument przyjąć selektor css, który filtruje znalezione elementy

- **find()** – znajduje elementy, które są zagnieżdżone w innym
- **closest()** – znajduje elementy najbliższe idąc w górę drzewa DOM
- **children()** – znajduje wszystkie dzieci danego elementu
- **parent()** – znajduje rodzica elementu
- **siblings()** – znajduje rodzeństwo elementu
- **next()** – znajduje następny element
- **prev()** – znajduje poprzedni element

```
$('div').find();  
$('div').find('p.head');  
$('div').parent();  
$('div').parent('li');  
$('div').siblings();  
$('div').siblings('span');  
$('div').next();  
$('div').next('h1');
```

Znajdzie element następny za każdym elementem **div**

Zaawansowane wyszukiwanie elementów

Metody do wyszukiwania elementów

Wszystkie te metody mogą jako argument przyjąć selektor css, który filtruje znalezione elementy

- **find()** – znajduje elementy, które są zagnieżdżone w innym
- **closest()** – znajduje elementy najbliższe idąc w górę drzewa DOM
- **children()** – znajduje wszystkie dzieci danego elementu
- **parent()** – znajduje rodzica elementu
- **siblings()** – znajduje rodzeństwo elementu
- **next()** – znajduje następny element
- **prev()** – znajduje poprzedni element

```
$( 'div' ).find();  
$( 'div' ).find( 'p.head' );  
$( 'div' ).parent();  
$( 'div' ).parent( 'li' );  
$( 'div' ).siblings();  
$( 'div' ).siblings( 'span' );  
$( 'div' ).next();  
$( 'div' ).next( 'h1' );
```

Znajdzie element następny o ile jest nim element **h1** za każdym elementem **div**

Zadania

Czas na zadania

Eventy

Eventy w jQuery

jQuery znacznie usprawniło korzystanie z eventów. Do sterowania eventami wystarczą trzy metody:

- **on(event, function)** – pozwala na dodanie callbacka do eventu,
- **one(event, function)** – pozwala na zapięcie nowego eventu, który zadziała tylko i wyłącznie raz, po czym zostanie automatycznie usunięty,

- **off(event, function)** – usuwa wszystkie callbacki, które były podpięte pod dany event (nawet te anonimowe).
- Nazwy eventów, które przeglądarka nam udostępnia, są w większości identyczne jak te wykorzystujące metodę **addEventListener**.

Propagacja eventów

Tak samo jak w czystym JavaScript eventy są propagowane. Jesteśmy w stanie zatrzymać propagację eventu, jeżeli użyjemy jednej z następujących metod:

- **stopPropagation()** – zatrzymuje propagację eventu w górę drzewa DOM (callbacki rodziców nie zostaną uruchomione).
- **stopImmediatePropagation()** – zatrzymuje propagację eventu oraz każdy inny event, który powinien zostać uruchomiony.

preventDefault()

Możemy zapobiec domyślnej akcji eventu np.

```
$( 'a' ).on( 'click', function() {  
    /* jakaś akcja po kliknięciu,  
    np. przeniesienie pod adres znajdujący się  
    w href zostanie normalnie wykonana */  
});  
  
$( 'a' ).on( 'click', function(event) {  
    event.preventDefault();  
    /* jakaś akcja po kliknięciu,  
    np. przeniesienie pod adres znajdujący się  
    w href zostanie anulowana */  
});
```

preventDefault() vs return false

Zapobiega domyślnej akcji eventu np.:

```
$( 'a' ).on( 'click', function(event) {  
    event.preventDefault();  
});
```

Zapobiega domyślnej akcji eventu oraz zapobiega propagacji eventu w górę.

```
$( 'a' ).on( 'click', function() {  
    return false;  
});
```

Przetestuj zadanie:

<http://jsfiddle.net/CodersLab/cw7z5g9x>

preventDefault() vs return false

Zapobiega domyślnej akcji eventu np.:

```
$( 'a' ).on( 'click', function(event) {  
    event.preventDefault();  
});
```

Zapobiega domyślnej akcji eventu oraz zapobiega propagacji eventu w górę.

```
$( 'a' ).on( 'click', function() {  
    return false;  
});
```

Pamiętamy aby w sytuacji użycia **return** nie umieszczać dalej żadnego kodu, gdyż się on nie wykona.

Przetestuj zadanie:

<http://jsfiddle.net/CodersLab/cw7z5g9x>

on()

Metoda on zaczepla określony event do elementu jQuery.

```
$(elements).on(events [, selector] [, data], handler)
```

Gdzie:

- **events** to typ eventu, może być jeden lub więcej,
- **selector** – to opcjonalny parametr, określa selektory, na których możemy zaczepić event, a których np. nie ma jeszcze w dokumencie,
- **data** – to również opcjonalny parametr. Możemy przekazać do funkcji handler jakieś dane np. {foo: "bar"},
- **handler** – to funkcja, która zostanie wykonana w momencie wywołania eventu.

on()

1. Dla każdego elementu, który znajdziesz, ustaw event **click**.
2. Dla każdego elementu, który ma ustawiony event, zostaje przypisana funkcja anonimowa.
3. Funkcja ta zostanie wywołana dopiero wtedy, gdy event **click** zostanie wywołany.

```
$('.menu').find('li').on('click', function() {  
    /* jakaś akcja po kliknięciu */  
});
```

one()

Metoda one zaczepia określony event do elementu jQuery tylko raz.

```
$(elements).one(events [, selector] [, data], handler);
```

Metoda przyjmuje te same parametry co **on()**.

```
$('.menu').find('li').one('click', function() {  
    /* jakaś akcja po kliknięciu */  
});
```

off()

Metoda off odczepia określony event od elementu jQuery.

```
$(elements).off([ events ] [, selector] [, handler])
```

- Wszystkie parametry są opcjonalne.
- Wywołanie samego **\$(element).off()** usunie wszystkie eventy z elementu.

```
$('.menu').find('li').on('click', function() {  
    /* jakaś akcja po kliknięciu */  
});  
$('.menu').find('li').off('click');  
//hint - można też  
var menuLi = $('.menu').find('li');  
menuLi.on('click', function() {  
    /* jakaś akcja po kliknięciu */  
});  
menuLi.off('click');
```

off()

Metoda off odczepia określony event od elementu jQuery.

```
$(elements).off([ events ] [, selector] [, handler])
```

- Wszystkie parametry są opcjonalne.
- Wywołanie samego **\$(element).off()** usunie wszystkie eventy z elementu.

```
$('.menu').find('li').on('click', function() {  
    /* jakaś akcja po kliknięciu */  
});
```

```
$('.menu').find('li').off('click');
```

//hint - można też

```
var menuLi = $('.menu').find('li');
```

```
menuLi.on('click', function() {  
    /* jakaś akcja po kliknięciu */  
});
```

```
menuLi.off('click');
```

Przypisujemy kolekcję znalezionych elementów do zmiennej aby nie odpytywać kolejny raz drzewa DOM

Metoda one – przykład

```
<button id="ourButton">Click me!</button>
```

```
$( '#ourButton' ).one( 'click', function (event) {  
    alert( 'You clicked me!' );  
});
```

Metoda one – przykład

```
<button id="ourButton">Click me!</button>
```

```
$( '#ourButton' ).one( 'click', function (event) {  
    alert( 'You clicked me!' );  
} );
```

(click - 1) 'You clicked me!'

Metoda one – przykład

```
<button id="ourButton">Click me!</button>
```

```
$( '#ourButton' ).one( 'click', function (event) {  
    alert( 'You clicked me!' );  
} );
```

(click - 2) Nic się nie wyświetla, ponieważ użyliśmy metody **one**

Metoda on – przykład

```
<button id="ourButton">Click me!</button>
```

```
$( '#ourButton' ).on( 'click', function (event) {  
    alert( 'You clicked me!' );  
});
```


Metoda on – przykład

```
<button id="ourButton">Click me!</button>
```

```
$( '#ourButton' ).on( 'click', function (event) {  
    alert( 'You clicked me!' );  
});
```

(click - 1) 'You clicked me!'

Metoda on – przykład

```
<button id="ourButton">Click me!</button>
```

```
$( '#ourButton' ).on( 'click', function (event) {  
    alert( 'You clicked me!' );  
});
```

(click - 2) 'You clicked me!'

Metoda on – przykład

```
<button id="ourButton">Click me!</button>
```

```
$( '#ourButton' ).on( 'click', function (event) {  
    alert( 'You clicked me!' );  
});
```

(click - n) 'You clicked me!'

Metoda off – przykład

```
<button id="ourButton">Click me!</button>
```

```
$( '#ourButton' ).on( 'click', function (event) {  
    alert( 'You clicked me!' );  
});  
$( '#ourButton' ).off( 'click' );
```

Metoda off – przykład

```
<button id="ourButton">Click me!</button>
```

```
$( '#ourButton' ).on( 'click', function (event) {  
    alert( 'You clicked me!' );  
});  
$( '#ourButton' ).off( 'click' );
```

(click - 1) Nic się nie wyświetla ponieważ odpięliśmy event.

Najpopularniejsze eventy

Mouse Events

- **click** – kliknięcie
- **dblclick** – podwójne kliknięcie
- **mouseenter** – najechanie
- **mouseleave** – zjechanie

KeyBoard Events

- **keypress** – wciśnięty klawisz
- **keydown** – wciśnięty klawisz (działa na klawisze specjalne)
- **keyup** – zwolniony klawisz

Form Events

- **submit** – kliknięty submit
- **change** – zmiana elementu
- **focus** – focus na elemencie
- **blur** – utrata eventu focus

Document/Window Events

- **load** – ładowanie dokumentu
- **resize** – zmiana wielkości okna
- **unload** – event po opuszczeniu przez użytkownika strony (blokowany przez niektóre przeglądarki, możesz użyć **onbeforeunload**)
- **scroll** – scrollowanie

Zadania

Czas na zadania

Manipulacja DOM-em

Atrybuty i własności

- **attributes (atrybuty)** - występują w tekstowym dokumencie HTML, są widoczne w źródle strony
- **properties (własności)** - są dostępne tylko przez JavaScript .

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet"
          href="css/style.css">
</head>
<body>
<a href="http://coderslab.pl">CL</a>

</body>
</html>
```

Atrybuty i własności

- **attributes (atrybuty)** - występują w tekstowym dokumencie HTML, są widoczne w źródle strony
- **properties (własności)** - są dostępne tylko przez JavaScript .

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet"
          href="css/style.css">
</head>
<body>
<a href="http://coderslab.pl">CL</a>

</body>
</html>
```

rel i **href** to atrybuty html

Atrybuty i własności

- **attributes (atrybuty)** - występują w tekstowym dokumencie HTML, są widoczne w źródle strony
- **properties (własności)** - są dostępne tylko przez JavaScript .

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet"
          href="css/style.css">
</head>
<body>
<a href="http://coderslab.pl">CL</a>

</body>
</html>
```

src, alt i title to atrybuty html

Atrybuty

Możemy łatwo wczytywać lub modyfikować atrybuty elementów dzięki następującym metodom:

- **attr(name, newValue)** – pobierz lub ustaw atrybut elementu,
- **removeAttr(name)** – usuń atrybut elementu,

Każda z tych metod może nastawiać atrybut (jeżeli podamy nową wartość) **attr(name, newValue)** lub zwracać jego wartość (jeżeli podamy tylko nazwę) **attr(name)** .

Kod HTML

```
<div class="footer"
      id="plan">0 nas</div>
```

Kod Javascript

```
var elementID = $('.footer').attr('id');
```

Kod Javascript

```
var elementID =
$('.footer').removeAttr('id');
```

Atrybuty

Możemy łatwo wczytywać lub modyfikować atrybuty elementów dzięki następującym metodom:

- **attr(name, newValue)** – pobierz lub ustaw atrybut elementu,
- **removeAttr(name)** – usuń atrybut elementu,

Każda z tych metod może nastawiać atrybut (jeżeli podamy nową wartość) **attr(name, newValue)** lub zwracać jego wartość (jeżeli podamy tylko nazwę) **attr(name)** .

Kod HTML

```
<div class="footer"
      id="plan">0 nas</div>
```

Kod Javascript

```
var elementID = $('.footer').attr('id');
```

"plan"

Kod Javascript

```
var elementID =
$('.footer').removeAttr('id');
```

Atrybuty

Możemy łatwo wczytywać lub modyfikować atrybuty elementów dzięki następującym metodom:

- **attr(name, newValue)** – pobierz lub ustaw atrybut elementu,
- **removeAttr(name)** – usuń atrybut elementu,

Każda z tych metod może nastawiać atrybut (jeżeli podamy nową wartość) **attr(name, newValue)** lub zwracać jego wartość (jeżeli podamy tylko nazwę) **attr(name)** .

Kod HTML

```
<div class="footer"
      id="plan">0 nas</div>
```

Kod Javascript

```
var elementID = $('.footer').attr('id');
```

Kod Javascript

```
var elementID =
$('.footer').removeAttr('id');
```

```
<div class="footer">0 nas</div>
```


Metoda prop()

- **prop** – sprawdza własności (properties) elementu.
- Jest używany podczas pobierania atrybutów boolean oraz własności nieistniejących w dokumencie HTML.
- Wszystkie inne atrybuty powinno się pobierać za pomocą **attr()**.

Przykład zastosowania tej metody:

<http://jsfiddle.net/bipen/54nLM/>

Kod HTML

```
<input type="checkbox" value="test"
      id="test"/>
```

Kod Javascript

```
$( '#test' ).prop( 'checked' );
$( '#test' ).prop( 'id' );
$( '#test' ).attr( 'id' );
```

Metoda prop()

- **prop** – sprawdza własności (properties) elementu.
- Jest używany podczas pobierania atrybutów boolean oraz własności nieistniejących w dokumencie HTML.
- Wszystkie inne atrybuty powinno się pobierać za pomocą **attr()**.

Przykład zastosowania tej metody:
<http://jsfiddle.net/bipen/54nLM/>

Kod HTML

```
<input type="checkbox" value="test"
      id="test"/>
```

Kod Javascript

```
$( '#test' ).prop( 'checked' );
$( '#test' ).prop( 'id' );
$( '#test' ).attr( 'id' );
```

Pobiera własność **id** z JavaScript

Metoda prop()

- **prop** – sprawdza własności (properties) elementu.
- Jest używany podczas pobierania atrybutów boolean oraz własności nieistniejących w dokumencie HTML.
- Wszystkie inne atrybuty powinno się pobierać za pomocą **attr()**.

Przykład zastosowania tej metody:

<http://jsfiddle.net/bipen/54nLM/>

Kod HTML

```
<input type="checkbox" value="test"
      id="test"/>
```

Kod Javascript

```
$( '#test' ).prop( 'checked' );
$( '#test' ).prop( 'id' );
$( '#test' ).attr( 'id' );
```

Pobiera atrybut html **id**

Metoda prop()

- **prop** – sprawdza własności (properties) elementu.
- Jest używany podczas pobierania atrybutów boolean oraz własności nieistniejących w dokumencie HTML.
- Wszystkie inne atrybuty powinno się pobierać za pomocą **attr()**.

Przykład zastosowania tej metody:
<http://jsfiddle.net/bipen/54nLM/>

Kod HTML

```
<input type="checkbox" value="test"
      id="test"/>
```

Kod Javascript

```
$( '#test' ).prop( 'checked' );
$( '#test' ).prop( 'id' );
$( '#test' ).attr( 'id' );
```

Są to dokładnie te same wartości tylko pobrane na 2 różne sposoby.

Atrybut data

Możemy nastawiać lub odczytywać **atrybut** za pomocą metody **data(dataSet, value)**.

Kod HTML

```
<div data-role="page"  
      data-last-value="43"  
      data-hidden="true">
```

Kod Javascript

```
$("div").data("role") ;  
$("div").data("lastValue");  
$("div").data("hidden");  
$("div").data("options", "new option" );
```

Atrybut data

Możemy nastawiać lub odczytywać **atrybut** za pomocą metody **data(dataSet, value)**.

Kod HTML

```
<div data-role="page"  
      data-last-value="43"  
      data-hidden="true">
```

Kod Javascript

```
$( "div" ).data( "role" ) ;  
$( "div" ).data( "lastValue" );  
$( "div" ).data( "hidden" );  
$( "div" ).data( "options", "new option" );
```

"page"

Atrybut data

Możemy nastawiać lub odczytywać **atrybut** za pomocą metody **data(dataSet, value)**.

Kod HTML

```
<div data-role="page"  
      data-last-value="43"  
      data-hidden="true">
```

Kod Javascript

```
$("div").data("role") ;  
$("div").data("lastValue");  
$("div").data("hidden");  
$("div").data("options", "new option" );
```

43

Atrybut data

Możemy nastawiać lub odczytywać **atrybut** za pomocą metody **data(dataSet, value)**.

Kod HTML

```
<div data-role="page"  
      data-last-value="43"  
      data-hidden="true">
```

Kod Javascript

```
$("div").data("role") ;  
$("div").data("lastValue");  
$("div").data("hidden");  
$("div").data("options", "new option" );
```

true

Pobieranie i wstawianie tekstu do elementu

- **html()** – wstawia/ustawia tekst lub HTML (zrenderowany)
- **text()** – wstawia/ustawia tekst lub HTML (jako string, np. *Tekst*)

Zobacz różnicę:

<http://jsfiddle.net/hossain/sUTVg/>

Modyfikowanie elementów

Tworzenie nowych elementów

- Tworzenie nowych elementów w jQuery staje się bardzo proste.
- Wystarczy, że string z kodem HTML danego elementu otoczymy `$()` i dostaniemy wrapper z tym elementem.
- Musimy pamiętać, że tak jak w przypadku czystego JavaScript, element ten nie będzie jeszcze podczepiony do DOM-u.

```
var newDiv = $("<div>");  
var newDiv =  
  $("<div>Lorem ipsum</div>");  
var newDiv =  
  $('<div class="foo bar" id="newDiv">');  
var newDiv =  
  $("<div>", { id: "myId",  
                class: "class1 class2"  
              });
```

Tworzenie nowych elementów

- Tworzenie nowych elementów w jQuery staje się bardzo proste.
- Wystarczy, że string z kodem HTML danego elementu otoczymy `$()` i dostaniemy wrapper z tym elementem.
- Musimy pamiętać, że tak jak w przypadku czystego JavaScript, element ten nie będzie jeszcze podczepiony do DOM-u.

```
var newDiv = $("<div>");  
var newDiv =  
  $("<div>Lorem ipsum</div>");  
var newDiv =  
  $('<div class="foo bar" id="newDiv">');  
var newDiv =  
  $("<div>", { id: "myId",  
                class: "class1 class2"  
                });
```

Podajemy tylko tag otwierający
<div></div>

Tworzenie nowych elementów

- Tworzenie nowych elementów w jQuery staje się bardzo proste.
- Wystarczy, że string z kodem HTML danego elementu otoczymy `$()` i dostaniemy wrapper z tym elementem.
- Musimy pamiętać, że tak jak w przypadku czystego JavaScript, element ten nie będzie jeszcze podczepiony do DOM-u.

```
var newDiv = $("<div>");  
var newDiv =  
    $("<div>Lorem ipsum</div>");  
var newDiv =  
    $('<div class="foo bar" id="newDiv">');  
var newDiv =  
    $("<div>", { id: "myId",  
                  class: "class1 class2"  
                });
```

Podajemy tag wraz z zawartością
`<div>Lorem ipsum</div>`

Tworzenie nowych elementów

- Tworzenie nowych elementów w jQuery staje się bardzo proste.
- Wystarczy, że string z kodem HTML danego elementu otoczymy `$()` i dostaniemy wrapper z tym elementem.
- Musimy pamiętać, że tak jak w przypadku czystego JavaScript, element ten nie będzie jeszcze podczepiony do DOM-u.

```
var newDiv = $("<div>");  
var newDiv =  
  $("<div>Lorem ipsum</div>");  
var newDiv =  
  $('<div class="foo bar" id="newDiv">');  
var newDiv =  
  $("<div>", { id: "myId",  
                class: "class1 class2"  
              });
```

Podajemy tag otwierający z atrybutami, bez zawartości

```
<div class="foo bar" id="newDiv">  
</div>
```

Tworzenie nowych elementów

- Tworzenie nowych elementów w jQuery staje się bardzo proste.
- Wystarczy, że string z kodem HTML danego elementu otoczymy `$()` i dostaniemy wrapper z tym elementem.
- Musimy pamiętać, że tak jak w przypadku czystego JavaScript, element ten nie będzie jeszcze podczepiony do DOM-u.

```
var newDiv = $("<div>");  
var newDiv =  
  $("<div>Lorem ipsum</div>");  
var newDiv =  
  $('<div class="foo bar" id="newDiv">');  
var newDiv =  
  $("<div>", { id: "myId",  
                class: "class1 class2"  
                });
```

Podajemy tag, i jako drugi argument listę atrybutów html w notacji obiektowej

```
<div id="myId" class="class1  
class2"></div>
```

Dodawanie elementów do DOM-u

- Tak samo jak w przypadku czystego JavaScript po utworzeniu elementu należy go jeszcze podpiąć do DOM-u.
- jQuery udostępnia nam bardzo dużo metod, dzięki którym możemy łatwo podpiąć element w wybranym miejscu.

Są to np.:

- after,
- before,
- append,
- appendTo,
- prepend,
- prependTo,
- insertAfter,
- insertBefore,
- wrap.

Wstawianie elementu przed lub po

- Oto metody, które służą do wstawiania elementów bezpośrednio przed wybranym elementem lub po nim:
 - **before()**
 - **after()**

```
<p class="bar">Hello</p>
```

```
var firstOfBar = $(".bar").first();  
var newElement = $('<div class="new">This is new element</div>');  
firstOfBar.after(newElement);  
firstOfBar.before(newElement);
```

Wstawianie elementu przed lub po

- Oto metody, które służą do wstawiania elementów bezpośrednio przed wybranym elementem lub po nim:
 - **before()**
 - **after()**

```
<p class="bar">Hello</p>
```

```
var firstOfBar = $(".bar").first();  
var newElement = $('<div class="new">This is new element</div>');  
firstOfBar.after(newElement);  
firstOfBar.before(newElement);
```


Wstawianie elementu przed lub po

- Oto metody, które służą do wstawiania elementów bezpośrednio przed wybranym elementem lub po nim:
 - **before()**
 - **after()**

```
<p class="bar">Hello</p>
```

```
var firstOfBar = $(".bar").first();  
var newElement = $('<div class="new">This is new element</div>');  
firstOfBar.after(newElement);  
firstOfBar.before(newElement);
```

```
<p class="bar">Hello</p>  
<div class="new">This is new element</div>
```

Wstawianie elementu przed lub po

- Oto metody, które służą do wstawiania elementów bezpośrednio przed wybranym elementem lub po nim:
 - **before()**
 - **after()**

```
<p class="bar">Hello</p>
```

```
var firstOfBar = $(".bar").first();  
var newElement = $('<div class="new">This is new element</div>');  
firstOfBar.after(newElement);  
firstOfBar.before(newElement);
```

```
<div class="new">This is new element</div>  
<p class="bar">Hello</p>
```

Wstawianie elementu przed lub po

- Dużo metod w jQuery ma swoje lustrzane odpowiedniki.
- Odpowiedniki **before()** i **after()** są następujące:
 - **insertBefore()**
 - **insertAfter()**

```
<p class="bar">Hello</p>
```

```
var firstOfBar = $(".bar").first();  
var newElement = $('<div class="new">This is new element</div>');  
newElement.insertAfter(firstOfBar);  
newElement.insertBefore(firstOfBar);
```

Wstawianie elementu przed lub po

- Dużo metod w jQuery ma swoje lustrzane odpowiedniki.
- Odpowiedniki **before()** i **after()** są następujące:
 - **insertBefore()**
 - **insertAfter()**

```
<p class="bar">Hello</p>
```

```
var firstOfBar = $(".bar").first();  
var newElement = $('<div class="new">This is new element</div>');  
newElement.insertAfter(firstOfBar);  
newElement.insertBefore(firstOfBar);
```

Wstawianie elementu przed lub po

- Dużo metod w jQuery ma swoje lustrzane odpowiedniki.
- Odpowiedniki **before()** i **after()** są następujące:
 - **insertBefore()**
 - **insertAfter()**

```
<p class="bar">Hello</p>
```

```
var firstOfBar = $(".bar").first();  
var newElement = $('<div class="new">This is new element</div>');  
newElement.insertAfter(firstOfBar);  
newElement.insertBefore(firstOfBar);
```

```
<p class="bar">Hello</p>  
<div class="new">This is new element</div>
```

Wstawianie elementu przed lub po

- Dużo metod w jQuery ma swoje lustrzane odpowiedniki.
- Odpowiedniki **before()** i **after()** są następujące:
 - **insertBefore()**
 - **insertAfter()**

```
<p class="bar">Hello</p>
```

```
var firstOfBar = $(".bar").first();  
var newElement = $('<div class="new">This is new element</div>');  
newElement.insertAfter(firstOfBar);  
newElement.insertBefore(firstOfBar);
```

```
<div class="new">This is new element</div>  
<p class="bar">Hello</p>
```

after() vs insertAfter()

Różnica polega na tym, że funkcja **insertAfter()** zwraca wszystkie znalezione elementy, natomiast funkcja **after** – nie zwraca nic.

```
$( "p" ).insertAfter( "#foo" );  
$( "#foo" ).after( $( "<p>" ) );
```

after() vs insertAfter()

Różnica polega na tym, że funkcja **insertAfter()** zwraca wszystkie znalezione elementy, natomiast funkcja **after** – nie zwraca nic.

```
$( "p" ).insertAfter( "#foo" );  
$( "#foo" ).after( $( "<p>" ) );
```

Obie linie zrobią to samo

Dodawanie elementu do dzieci

Możemy łatwo dodać element do dzieci innego elementu dzięki następującym metodom:

- **append(newElement)** – wstaw nowy element na koniec dzieci już istniejącego elementu,
- **appendTo(oldElement)** – odwrotność (czyli wywołujemy na nowym elemencie i podajemy, gdzie ma się dodać).

- **prepend(newElement)** – wstaw nowy element na początek dzieci już istniejącego elementu,
- **prependTo(oldElement)** – odwrotność.

append vs prepend

Kod HTML

```
<div class="foo" id="fooId" style="color: red;">  
  <p class="bar">Hello</p>  
</div>
```

Kod Javascript

```
var newElement = $('<div class="new">This is new element</div>');  
var foo = $("#fooId");  
foo.append(newElement);  
newElement.appendTo(foo);
```

append vs prepend

Kod HTML

```
<div class="foo" id="fooId" style="color: red;">  
  <p class="bar">Hello</p>  
</div>
```

Kod Javascript

```
var newElement = $('<div class="new">This is new element</div>');  
var foo = $("#fooId");  
foo.append(newElement);  
newElement.appendTo(foo);
```

Obie linie dadzą ten sam wynik w przeglądarce

```
<div class="foo" id="fooId" style="color: red;">  
  <p class="bar">Hello</p>  
  <div class="new">This is new element</div>  
</div>
```

append vs prepend

Kod HTML

```
<div class="foo">  
  <p class="bar">Hello</p>  
</div>
```

Kod Javascript

```
var newElement = $('<div class="new">This is new element</div>');  
var foo = $("#fooId");  
foo.prepend(newElement);  
newElement.prependTo(foo);
```

append vs prepend

Kod HTML

```
<div class="foo">  
  <p class="bar">Hello</p>  
</div>
```

Kod Javascript

```
var newElement = $('<div class="new">This is new element</div>');  
var foo = $("#fooId");  
foo.prepend(newElement);  
newElement.prependTo(foo);
```

Obie linie dadzą ten sam wynik w przeglądarce

```
<div class="foo" id="fooId" style="color: red;">  
  <div class="new">This is new element</div>  
  <p class="bar">Hello</p>  
</div>
```

Usuwanie elementów z DOM-u

Oto kilka metod ułatwiających usunięcie elementów z DOM-u. Co ważne, jedna z metod pozwala nam odzepić element bez jego niszczenia.

- **remove()** – usuń element,
- **detach()** – wypnij element z drzewa DOM bez usuwania go i zwróć go (np. żebyśmy mogli zapisać go do zmiennej),
- **empty()** – usuń wszystko ze środka elementu

Usuwanie elementów z DOM-u

Kod HTML

```
<div class="foo">
  <p class="bar1">Hello1</p>
  <p class="bar2">Hello2</p>
  <p class="bar3">Hello3</p>
</div>
```

Kod JavaScript

```
$( ".bar1" ).remove();
```

Wynik

```
<div class="foo">
  <p class="bar2">Hello2</p>
  <p class="bar3">Hello3</p>
</div>
```

Usuwanie elementów z DOM-u

Kod HTML

```
<div class="foo">
  <p class="bar1">Hello1</p>
  <p class="bar2">Hello2</p>
  <p class="bar3">Hello3</p>
</div>
```

Kod JavaScript

```
$(".bar1").remove();
```

Wynik

```
<div class="foo">
  <p class="bar2">Hello2</p>
  <p class="bar3">Hello3</p>
</div>
```

Paragraf zostanie usunięty

Usuwanie elementów z DOM-u

Kod HTML

```
<div class="foo">
  <p class="bar2">Hello2</p>
  <p class="bar3">Hello3</p>
</div>
```

Kod JavaScript

```
var removedBar2 = $(".bar2").detach();
```

Wynik

```
<div class="foo">
  <p class="bar3">Hello3</p>
</div>
```

Usuwanie elementów z DOM-u

Kod HTML

```
<div class="foo">  
  <p class="bar2">Hello2</p>  
  <p class="bar3">Hello3</p>  
</div>
```

Kod JavaScript

```
var removedBar2 = $(".bar2").detach();
```

Wynik

```
<div class="foo">  
  <p class="bar3">Hello3</p>  
</div>
```

Paragraf zostanie wypięty, można z nim coś zrobić ponieważ jest w zmiennej

Usuwanie elementów z DOM-u

Kod HTML

```
<div class="foo">  
  <p class="bar3">Hello3</p>  
</div>
```

Kod JavaScript

```
$(".foo").empty();  
//można porównać do $(".foo").html("");
```

Wynik

```
<div class="foo"></div>
```

Usuwanie elementów z DOM-u

Kod HTML

```
<div class="foo">  
  <p class="bar3">Hello3</p>  
</div>
```

Kod JavaScript

```
$(".foo").empty();  
//można porównać do $(".foo").html("");
```

Wynik

```
<div class="foo"></div>
```

Usuwamy zawartość elementu **div**

Eventy zaawansowane

Eventy zaawansowane

Dodając event do elementów wyszukanych selektorem, np. elementy listy, eventy podpięte są jedynie do tych elementów, które istnieją w DOM w momencie podpięcia eventu.

- Elementy dodane dynamicznie nie zareagują na event
- Musimy event założyć na dowolnego istniejącego przodka (np. rodzica) i przekazać metodzie `on('click')` jako dodatkowy argument selektor elementu jakiego event nas interesuje.

Eventy zaawansowane

```
<ul id="shoppingList">
  <li class="product_1">Tomato</li><!-- event -->
  <li class="product_2">Potato</li><!-- event -->
  <li class="product_3">Salt</li><!-- event -->
  <li class="product_4">Carrot</li><!-- event -->
</ul>
```

```
var shoppingList = $('#shoppingList');
var products = shoppingList.find('li');
products.on('click', function(){
  console.log('product clicked');
});
```

Eventy zaawansowane

```
<ul id="shoppingList">
  <li class="product_1">Tomato</li><!-- event -->
  <li class="product_2">Potato</li><!-- event -->
  <li class="product_3">Salt</li><!-- event -->
  <li class="product_4">Carrot</li><!-- event -->
</ul>
```

```
var shoppingList = $('#shoppingList');
var products = shoppingList.find('li');
products.on('click', function(){
  console.log('product clicked');
});
```

Event został dodany na 4 elementy **li**

Eventy zaawansowane

Założmy, że do naszej strony dodaliśmy 2 elementy dynamicznie np. przez jQuery

```
<ul id="shoppingList">
  <li class="product_1">Tomato</li><!-- event -->
  <li class="product_2">Potato</li><!-- event -->
  <li class="product_3">Salt</li><!-- event -->
  <li class="product_4">Carrot</li><!-- event -->
  <li class="product_5">Carrot</li>
  <li class="product_6">Carrot</li>
</ul>
```

Eventy zaawansowane

Założmy, że do naszej strony dodaliśmy 2 elementy dynamicznie np. przez jQuery

```
<ul id="shoppingList">
  <li class="product_1">Tomato</li><!-- event -->
  <li class="product_2">Potato</li><!-- event -->
  <li class="product_3">Salt</li><!-- event -->
  <li class="product_4">Carrot</li><!-- event -->
  <li class="product_5">Carrot</li>
  <li class="product_6">Carrot</li>
</ul>
```

Nowo dodane elementy **li** nie mają dodanego eventu, ponieważ jQuery nie знаło tych elementów na moment dodawania eventu

Eventy zaawansowane

```
<ul id="shoppingList"><!-- event -->
  <li class="product_1">Tomato</li><!-- dziecko propagujące -->
  <li class="product_2">Potato</li><!-- dziecko propagujące -->
  <li class="product_3">Salt</li><!-- dziecko propagujące -->
  <li class="product_4">Carrot</li><!-- dziecko propagujące -->
  <li class="product_5">Carrot</li><!-- dziecko propagujące -->
  <li class="product_6">Carrot</li><!-- dziecko propagujące -->
</ul>
```

```
var shoppingList = $('#shoppingList');
shoppingList.on('click', 'li', function(){
  console.log('product clicked');
});
```

Eventy zaawansowane

```
<ul id="shoppingList"><!-- event -->
  <li class="product_1">Tomato</li><!-- dziecko propagujące -->
  <li class="product_2">Potato</li><!-- dziecko propagujące -->
  <li class="product_3">Salt</li><!-- dziecko propagujące -->
  <li class="product_4">Carrot</li><!-- dziecko propagujące -->
  <li class="product_5">Carrot</li><!-- dziecko propagujące -->
  <li class="product_6">Carrot</li><!-- dziecko propagujące -->
</ul>
```

```
var shoppingList = $('#shoppingList');
shoppingList.on('click', 'li', function(){
  console.log('product clicked');
});
```

Aby nasz event zadziałał zawsze, dodajemy go na element rodzica, który od początku istnieje w DOM, tutaj na listę, a następnie jako drugi argument podajemy selektor elementu, którego event ma dotyczyć

Eventy zaawansowane

```
<ul id="shoppingList"><!-- event -->
  <li class="product_1">Tomato</li><!-- dziecko propagujące -->
  <li class="product_2">Potato</li><!-- dziecko propagujące -->
  <li class="product_3">Salt</li><!-- dziecko propagujące -->
  <li class="product_4">Carrot</li><!-- dziecko propagujące -->
  <li class="product_5">Carrot</li><!-- dziecko propagujące -->
  <li class="product_6">Carrot</li><!-- dziecko propagujące -->
</ul>
```

```
var shoppingList = $('#shoppingList');
shoppingList.on('click', 'li', function(){
  console.log('product clicked');
});
```

Po kliknięciu w dowolny element listy, zadziała propagacja eventu na rodzica. Do rodzica podpięty jest event, sprawdzi on tylko, dzięki drugiemu argumentowi czy element wywołujący event to **li**

Zadania

Czas na zadania