jQuery TABLICE INFORMATYCZNE

Włodzimierz Gajda

Helion

Dołączanie biblioteki jQuery do dokumentu HTML

```
// plik lokalny
<script type="text/javascript" src="jquery-1.6.2.min.js">
</script>
```

Atrybut `src` może wskazywać pliki na serwerach Google lub jQuery:

```
src="http://code.jquery.com/jquery-1.6.2.min.js"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.min.js"
```

W powyższych adresach możemy wymienić numer wersji biblioteki jQuery.

Osadzanie kodu jQuery w dokumencie HTML

```
<head>
    ...
    <script type="text/javascript" src="jquery-
    1.6.2.min.js">
    </script>
    <script type="text/javascript">
    $(document).ready(function(){
        // kod wykorzystujący jQuery
    });
    </script>
</head>
```

Skrypty wykorzystujące jQuery możemy dołączać do strony WWW wielokrotnie:

```
<script type="text/javascript">
$(document).ready(function(){
    // pierwszy skrypt
});
</script>
<script type="text/javascript">
$(document).ready(function(){
    // drugi skrypt
});
</script>
```

Zapisywanie kodu jQuery w osobnym pliku .js

```
// plik index.html
<head>
    ...
    <script type="text/javascript" src="jquery-
    1.6.2.min.js">
    </script>
    <script type="text/javascript" src="skrypt.js">
    </script>
</head>
// plik skrypt.js
```

```
$(document).ready(function(){
    // kod wykorzystujący jQuery
});
```

Zapisywanie kodu JavaScript w osobnym pliku o rozszerzeniu *.js* jest najlepszą metodą dołączania skryptów JS do stron WWW; treść skryptów *.js* nie ma wtedy wpływu na poprawność kodów HTML.

Wykonywanie kodu JS po utworzeniu drzewa DOM

```
// zapis pełny
$(document).ready(function(){
    // kod wykonywany po utworzeniu drzewa DOM
});
```

```
// zapis skrócony
$(function(){
    // kod wykonywany po utworzeniu drzewa DOM
});
```

Wstrzymanie wykonywania zdarzenia ready

Funkcja `holdReady()` wywołana z parametrem `true` wstrzymuje generowanie zdarzenia `ready`. Gwarantuje to, że wykonanie zdarzenia `ready` nastąpi dopiero po załadowaniu skryptu *myplugin.js*:

```
$.holdReady(true);
$.getScript("myplugin.js", function() {
    $.holdReady(false);
});
```

Funkcja jQuery() i alias $()

```
// zapis pełny
jQuery(...);
// zapis wykorzystujący alias
$(...);
```

Jeśli w skrypcie jQuery nie wywołano funkcji `noConflict()`, poniższe instrukcje są równoważne:

```
$('p').css('background', 'yellow');
jQuery('p').css('background', 'yellow');
```

Użycie jQuery w połączeniu z innymi bibliotekami wykorzystującymi alias $ (np. Prototype)

Funkcja `noConflict()` usuwa odnoszący się do `jQuery()` alias `$` z globalnej przestrzeni nazewniczej. Dzięki temu bibliotekę jQuery możemy łączyć z innymi bibliotekami JavaScript, stosującymi alias `$`.

```
jQuery.noConflict();
jQuery(function(){
    // kod wykorzystujący jQuery
    // funkcje jQuery są dostępne jako jQuery()
});
// kod wykorzystujący $ jako obiekt innej biblioteki
```

Przekazana do funkcji `jQuery()` funkcja anonimowa może przyjmować parametry. Jeśli w roli parametru użyjemy znaku `$`, możemy wykorzystywać alias `$` w zapisie kodu z jQuery:

```
jQuery.noConflict();
jQuery(function($){
    // kod wykorzystujący jQuery
    // funkcje jQuery są dostępne jako jQuery() oraz $()
});
// kod wykorzystujący $ jako obiekt innej biblioteki
```

Różne rodzaje parametrów funkcji jQuery()

Wywołanie	Parametr wywołania	Opis
`$(function(){ ... });`	Funkcja anonimowa.	Skrótowy zapis wywołania metody `ready()`: `$(document).ready(function(){ ... });`
`$('<p></p>')`	Kod HTML.	Tworzenie nowych węzłów DOM.
`$('p')` `$('#info')` `$('.menu')`	Selektor.	Tworzenie obiektu jQuery zawierającego zbiór węzłów DOM, pasujących do podanego selektora.
`var el = document.getElementById('tresc'); $(el).text('abc...');`	Węzeł DOM, np. wynik metody `getElementById()`.	Tworzenie obiektu jQuery zawierającego wskazany węzeł DOM.
`var elementy = document.getElementsByTagName('li'); $(elementy).append(' - ???');`	Tablica węzłów DOM, np. wynik metody `getElementsByTagName()`.	Tworzenie obiektu jQuery zawierającego zbiór podanych węzłów DOM.
`$(this).toggle();`	Wewnątrz metody obsługi danego zdarzenia `this` jest obiektem, którego zdarzenie oprogramowujemy.	Tworzenie obiektu jQuery zawierającego wskazany węzeł DOM.
`$(document.body).css('color', 'red'); $(formularz.elements).hide();`	Predefiniowane obiekty globalne JavaScript, np. `document`, `window`.	Tworzenie obiektu jQuery zawierającego wskazany węzeł DOM.
`var p1 = $('#p'); var p2 = $(p1);`	Obiekt jQuery.	Klonowanie: obiekt `p2` jest kopią obiektu `p1`; zbiór węzłów DOM tych obiektów będzie identyczny.
`$()`	Wywołanie bezparametrowe.	Tworzy obiekt jQuery reprezentujący zbiór pusty węzłów DOM.

Bez względu na rodzaj parametrów wynikiem funkcji `jQuery()` zawsze jest obiekt jQuery.

Dodatkowe parametry funkcji jQuery()

Kontekst

Jeśli pierwszym parametrem funkcji `jQuery()` jest selektor, jej dodatkowym parametrem może być węzeł drzewa DOM. Parametr ten, nazywany *kontekstem*, ustala gałąź drzewa DOM, w której nastąpi wyszukiwanie. Instrukcje:

```
var el = $('#tresc');
$('p', el).css('color', 'red');
```

zmienią kolor akapitów zawartych wewnątrz elementu `#tresc`.

Atrybuty HTML

Jeśli pierwszym parametrem funkcji `jQuery()` jest kod HTML, to dodatkowym parametrem może być tablica atrybutów HTML i właściwości jQuery. Instrukcja:

```
$('<p></p>', {
    "class": "info",
    "id" : "akapit",
    text: "Lorem ipsum...",
    css: {'color': 'blue'},
    click: function(){
        $(this).hide();
    }
}).appendTo('body');
```

tworzy akapit `p` o podanych właściwościach. Parametry ujęte w cudzysłów (m.in. `class` oraz `id`) są atrybutami HTML, a pozostałe (`css`, `text` oraz `click`) — właściwościami jQuery.

Uwaga: tworzony element musi być pusty. Poprawnymi przykładami użycia powyższej metody są wywołania:

```
$('<p></p>', {...});
$('<p/>', {...});
```

Instrukcja:

```
$('<p>abc</p>', {...});
```

spowoduje utworzenie elementu `p` zawierającego tekst `abc`. Atrybuty podane jako drugi parametr nie zostaną zastosowane.

Wynik działania funkcji jQuery()

Wynikiem funkcji `jQuery()` jest obiekt jQuery, który reprezentuje zbiór wybranych elementów, będących węzłami DOM. W celu sprawdzenia liczby elementów zbioru możemy użyć właściwości `length` oraz metody `size()`:

```
var ile = $('p').length;
var ile = $('div').size();
```

Numeracja zawartych w obiekcie jQuery węzłów DOM, zaczyna się od 0. Dostęp do poszczególnych węzłów zapewnia metoda `get()` oraz interfejs tablicowy. Oto instrukcje, które pozwolą na uzyskanie dostępu do czwartego akapitu i szóstego elementu `div`:

```
var akapit_nr_4 = $('p').get(3);
var div_nr_6 = $('div')[5];
```

Uwaga: zmienne `akapit_nr_4` oraz `div_nr_6` są węzłami drzewa DOM, a nie obiektami jQuery. W celu konwersji na obiekty jQuery należy je przekazać do metody `jQuery()`, np.:

```
$(akapit_nr_4).css('background', 'yellow');
$(div_nr_6).css('background', 'yellow');
```

Metoda/właściwość	Opis	Przykład
`size()`	Metoda zwracająca liczbę elementów zawartych w zbiorze jQuery.	`$('p').size();`
`length`	Właściwość zwracająca liczbę elementów zawartych w zbiorze jQuery.	`$('p').length;`
`get()`	Metoda zwracająca element o podanym indeksie.	`// trzeci element zbioru jQuery` `$('p').get(2);`
`odwołanie tablicowe []`	Odwołanie zwracające element o podanym indeksie.	`// piąty element zbioru jQuery` `$('p')[4];`

Łańcuchy wywołań (ang. fluent interface)

Metody obiektu jQuery (m.in. `css()`, `attr()`, `click()`, `mouseover()`, `hide()`) zwracają obiekt jQuery, dlatego możemy tworzyć łańcuchy wywołań:

```
$(...).css(...).attr(...).click(...).mouseover(...).hide(...);
```

W literaturze anglojęzycznej rozwiązanie takie określa się terminem *fluent interface*.

Uwaga: w przypadku definiowania obsługi zdarzenia `load`, tj. gdy parametrem funkcji `$()` jest funkcja anonimowa:

```
$(function(){..});
```

nie należy tworzyć łańcuchów wywołań:

```
// PRZYKŁAD BŁĘDNY
$(function(){..}).css(...).html(...);
```

Iteracja elementów zawartych w zbiorze jQuery

Obiekt jQuery reprezentuje zbiór węzłów DOM. Metody jQuery (np. `css()`, `addClass()`, `show()`) zazwyczaj przetwarzają wszystkie elementy zawarte w danym zbiorze. Instrukcja:

```
$('p').css('color', 'red');
```

wykonuje iterację przetwarzającą wszystkie elementy `p`. Powyższy zapis należy rozumieć analogicznie do zapisu CSS:

```
p {
    color: red;
}
```

Do wykonania jawnej iteracji elementów zbioru należy użyć metody `each()`, np.:

```
$('p').each(...);
```

a nie instrukcji `for`.

Modyfikacja zbioru

Niektóre metody jQuery nie modyfikują zbioru elementów. Przykładem takich metod są `attr()` oraz `css()`. Po wywołaniu:

```
var z1 = $('p');
var z2 = $('p').css(...).attr(...);
```

zmienne `z1` oraz `z2` reprezentują identyczne zbiory węzłów. Inne metody (np. `next()`, `prev()`, `first()`, `last()`, `add()`) zmieniają zbiór węzłów zawartych w obiekcie jQuery.

Przetwarzanie pojedynczego elementu

Niektóre metody jQuery operują na pojedynczym węźle DOM. Przykładem takiej metody jest `html()` — metoda zwracająca zawartość węzła. Przetwarzanie jest tu ograniczone do pierwszego elementu zbioru jQuery. Instrukcja:

```
var t = $('p').html();
```

umieszcza w zmiennej `t` kod HTML, pobrany z pierwszego akapitu. Przykład równoważny z poprzednim to:

```
var t = $('p').first().html();
```

Selektory

Znaki specjalne

Znaki specjalne występujące w selektorach:

```
!"#$%&'(),+,./:;<=>?@[\]^`{|}~
```

Cytowanie znaków specjalnych podwójnym znakiem `\\`:

```
$("#foo\\.bar")
```

Ciąg dalszy na stronie 2 ➔