

jQuery

Pisz mniej, rób więcej

Co to jest jQuery? Wikipedia podaje następującą definicję: lekka Biblioteka programistyczna dla języka JavaScript, ułatwiająca współdziałanie JavaScript oraz HTML. Ale to nie wszystko. Jest to biblioteka, która wszystkie obiekty strony HTML „ubiera” w dodatkowe zdarzenia, własności i metody.

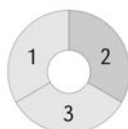
Dowiesz się...

- Artykuł pokazuje jak w łatwy i szybki sposób dodać do strony zaawansowane efekty wizualne, a także możliwości wykonywania asynchronicznych zapytań http.

Powinieneś wiedzieć...

- Wymagana jest dobra znajomość html, css,
- Podstawowa znajomość JavaScript, php.

Poziom trudności



Dość bardzo intuicyjne tworzy grupy obiektów i kolekcje, dając możliwość iteracji po wybranych elementach grupy. Wyposażenie jQuery w możliwość wykonywania zapytań asynchronicznych w połączeniu z łatwością manipulacji dowolnymi obiektami lub grupami obiektów wewnątrz strony HTML daje silne i łatwe w użyciu narzędzie do tworzenia atrakcyjnych wizualnie stron w technologii AJAX.

Instalacja i konfiguracja

Instalacja jest bardzo prosta, a właściwie nie ma jej wcale. Wystarczy ściągnąć plik jquery.js i włączyć go jako skrypt do własnej strony. Od tego momentu można zacząć używać funkcjonalności, jaką daje nam ta biblioteka. Pobieramy plik *jquery.js* z adresu: <http://code.jquery.com/jquery-latest.js> i umieszczamy go w katalogu roboczym razem z plikiem tworzonej przez nas strony. Następnie włączamy w kod strony zawartość ściągniętego pliku, deklarując w części nagłówkowej naszej strony:

```
<script src="jquery-latest.js" type="text/
  javascript"></script>
```

Tworzymy kod powodujący, iż po kliknięciu na dowolny link w obrębie strony HTML po-

jawi się komunikat (`alert`) „Hello World”. Ta konstrukcja odpowiada klasycznemu kodowi HTML i JavaScript

```
<a href="" onclick="alert('Hello
  world')">Link</a>
```

z tą różnicą, że jeden wpis odnosi się automatycznie do wszystkich linków na tworzonej przez nas stronie.

Selektory HTML i CSS

Pełne wsparcie dla CSS (1, 2, 3) oraz wszystkich selektorów HTML stanowi podstawę dobrego imienia omawianej biblioteki. Możliwość definicji kolekcji selektorów i iteracji po dowolnej z nich jest dużym ułatwieniem w tworzeniu logiki aplikacji webowej po stronie klienta. Przypuśćmy że chcemy odwołać się do elementu o ID „orderedlist”. W klasycznym skrypcie JavaScript musimy użyć konstrukcji: `document.getElementById("orderedlist")` by wydobyć interesujący nas element. Przy wsparciu jQuery mamy do dyspozycji konstrukcję:

```
$("#orderedlist").addClass("red");
```

Przy czym możemy na wybranym obiekcie wywołać dowolne funkcje dostarczane z omawianą biblioteką, w tym konkretnym przykładzie została wywołana funkcja

```
addClass("red");
```

Funkcja ta, jak łatwo się domyślić, dodaje klasę „red” do elementu o ID „orderedlist”.

Można się pokusić o stwierdzenie, że nie wydaje się to wielkim ułatwieniem, wszak JavaScript sam w sobie daje możliwość wyselekcjonowania wybranego elementu, ale dla konstrukcji, która ma za zadanie wybrać wszystkie elementy listy `` dla obiektu o wybranym ID, kod JavaScript będzie wymagał odrobiny programowania, a w przypadku jQuery wystarczy zastosować wyselekcjonowaną konstrukcję prezentowaną w listingu 8. Innym ciekawym przykładem selekcji elementów strony HTML może być ujęcie w kolekcję wszystkich odnośników mających ustawiony parametr „name”. Kod w jQuery napisać można w następującej postaci:

```
$("#a[@name]").css("background", "#eee"
  );
```

Łatwo się domyślić, że wykonanie tego fragmentu kodu spowoduje zmianę koloru tła wyselekcjonowanych elementów. Bardziej użytecznym selektorem jednak wydać się może taki, który wybiera linki zawierające w swym adresie wskazanie do np. katalogu „gallery”:

```
$("#a[@href*/gallery]").click(function() {
  // do something with all links that
  // point somewhere to /gallery
});
```

Można długo opisywać różne ciekawe kombinacje metod selekcji elementów strony, zamieszczone tutaj przykłady zostały wybrane, by pokazać spektrum możliwości, jakie na tym polu daje nam jQuery.

Walidacja formularzy

Częstym i bardzo eleganckim rozwiązaniem stosowanym na różnego rodzaju stronach jest walidacja formularzy. Nie trzeba chyba nikogo przekonywać, że duże formularze na

witrynach pełnych elementów graficznych lepiej walidować bez kosztownego przeladowywania całej strony, gdyż słabe łącze może spowodować, że wypełniający formularz zirytuje się długim oczekiwaniem na odpowiedź serwera z informacją, że wpisana wartość jest niewłaściwa. Zaczniemy od przykładu, w którym będziemy walidować numer PESEL, a w przypadku wpisania niewłaściwej wartości ostrzeżemy użytkownika, ale pozwolimy mu wypełniać dalej formularz i ostatecznie zaakceptujemy wadliwy numer PESEL. Najpierw przygotowujemy funkcję, która dokona sprawdzenia, czy przekazana jako argument wartość jest poprawnym numerem PESEL.

Część funkcji zmieniona w komentarz umożliwi w szybki sposób rozszerzenie funkcjonalność testu o sprawdzenie poprawności podanej w innym polu formularza płci, w tym celu należy dodać kolejny argument funkcji o nazwie „plec” (dozwolone wartości to M lub K) i usunąć znaki komentarza z odpowiedniego fragmentu kodu. Mając tak przygotowaną funkcję, możemy sprawić, by została wywołana przy wysłaniu formularza. Aby na nasze żądanie mimo błędnego wpisu dane zostały wysłane, zastosujemy jeszcze jedną funkcję pomocniczą. Teraz szybko powiążanie funkcji do zdarzenia wysłania formularza przy pomocy jQuery.

```
$('#frmDaneOsobowe').ajaxForm( {
    beforeSubmit: validate } );
```

Czegóż wybredny programista chciałby mieć więcej w tej sytuacji? Możliwość sprawdzenia, czy dany numer PESEL istnieje w naszej lokalnej bazie danych? Wywołanie asynchronicznego zapytania `get` i uzyskanie odpowiedzi od przygotowanego odpowiednio skryptu PHP wydaje się właściwe. W podobny sposób można sprawdzić adres e-mail i w czasie, gdy użytkownik wypełnia resztę formularza, możemy przesłać e-mail do przygotowanego skryptu PHP, który nie tylko sprawdzi poprawność adresu pod względem budowy (ciąg znaków, literka at, kolejny ciąg znaków), lecz także odpyta serwery DNS o rekordy MX czy nawet system pocztowy o poprawność adresu e-mail. Aby jednak to zrobić, zapoznajmy się z funkcjonalnością zapytań asynchronicznych oferowaną przez jQuery.

GET I POST asynchronicznie

To, co czyni z omawianej biblioteki uniwersalną do różnych zastosowań, to możliwość wysyłania asynchronicznych zapytań. Biblioteka ta ma gotowy zbiór funkcji wspierających zapytania asynchroniczne i – co bardzo użyteczne – funkcje te jako jeden z parametrów przyjmują tzw. *callback*, a dokładnie nazwę funkcji, która ma być wywołana na zakończenie

działania danego zapytania. Pierwszą użyteczną funkcją, związaną z dziedziną „*ajax*”, jest funkcja `get`. Parametrami jej wywołania są: `.get(url, params, callback)`, gdzie `url` to URL, `params` określa parametry przekazywane w zwykłym URL-u za znakiem zapytania, u nas będą w postaci:

```
{ name: "PHPSolution", id: "3/2007" }
```

Ostatnim parametrem może być nazwa funkcji, którą chcemy wywołać po wykonaniu zapytania, może być to funkcja formatująca treść stronę w zależności od uzyskanej odpowiedzi. Dla przykładu podamy kod, który po

załadowaniu danych ze skryptu PHP, załaduje je do znacznika HTML o ID = „wynik” (`<div id="wynik"></div>`)

W pliku *test.php* przetwarzamy zapytanie oraz przekazane parametry – Listing 11. Oczywiście, taki prosty przykład jest dobry, ale dla nas dobry to za mało, potrzebne nam są zapytania, które odpowiednio zareagują w przypadku niepowodzenia zapytania, a także – co być może ważniejsze – zrealizują odpowiednie czynności po pełnym wykonaniu zapytania, czyli wtedy, kiedy „spłyną” już wszystkie dane od serwera. Jest to potrzebne w przypadku przetwarzania przez serwer skomplikowanych operacji bazodanowych.

Listing 1. Funkcja sprawdzająca poprawność numeru PESEL

```
function validPESEL(pesel)
{
    if(pesel == 0) return true;
    if (pesel.length > 11) return false;
    var rePesel = /(\\d{11})/;

    if (rePesel.test(pesel)) pesel = RegExp.$1;
    else return false;
    pesel = pesel.split("");

    //if(plec)
    //{
    //    if(pesel[9]%2 && plec.search("K")!=-1) return false;
    //    else if (!pesel[9]%2 && plec.search("M")) return false;
    //}

    var wagi = new Array(1,3,7,9,1,3,7,9,1,3);
    var suma=0;

    for(i=0;i<=9;i++) suma += pesel[i]*wagi[i];
    suma %= 10;
    var sumaKontrolna = (10-suma) % 10;

    if (pesel[10]==sumaKontrolna) return true;
    else return false;
}
```

Listing 2. Formularz HTML z logowaniem asynchronicznym

```
<form method="get" class="cmxform" id="form" action="form.php">
  <fieldset>
  <p>
    <label for="login">Twój login</label>
    <input id="user" name="user" title="Podaj login, przynajmniej trzy znaki"
      class="required:true,minLength:3" />
  </p>
  <p>
    <label for="pass">Hasło</label>
    <input type="password" name="password" id="password" class="{
      required:true,minLength:5}" />
  </p>
  <p>
    <input class="submit" type="submit" value="Login"/>
  </p>
  </fieldset>
</form>
```

Listing 3. Wywołania funkcji dokonujących walidacji

```

<script type="text/javascript">
  jQuery(function() {

    // show loading indicator

    var loader = jQuery('<div id="loader"></div>')
      .css({position: "relative", top: "1em", left: "25em"})
      .hide()
      .appendTo("body");
    jQuery().ajaxStart(function() {
      loader.show();
    }).ajaxStop(function() {
      loader.hide();
    });

    jQuery().ajaxError(function(a, b, e) {
      throw e;
    });
    jQuery.validator.setDefaults({
      debug: true
    });

    var v = jQuery("#form").validate({
      submitHandler: function(form) {
        jQuery(form).ajaxSubmit({
          dataType: "json",
          after: function(result) {

            if(result.status) {
              v.showErrors(result.data);
              v.focusInvalid();
            }
          }
        });
      }
    });
    jQuery("#reset").click(function() {
      v.resetForm();
    });
  });
</script>

```

Listing 4. Przykładowy plik php symulujący logowanie użytkownika

```

<?php // oczekiwanie, w celu symulacji opóźnień w przetwarzaniu I przesyłaniu
// danych

sleep(1);
$user = $_REQUEST['user'];
$password = $_REQUEST['password'];

if($user && $password && $password == "pass")
    echo '{"status': 0, 'data': 'Cześć $user, Witamy ponownie.'}";

else

    echo '{"status': 1, 'data': {'password':
        'Przykro mi, Twoje hasło jest błędne.'}}';

?>

```

Taką funkcją jest: `$.ajax(params)`; Parametry przekazywane tej funkcji wymagają dokładniejszego omówienia.

Przed wszystkim forma przekazywanych parametrów jest w postaci par klucz: „wartość”. Nie wszystkie parametry są obowiązkowe, a te, które nie będą ustawione, zostaną zastąpione wartościami domyślnymi. Pierwszym niezbędnym parametrem jest URL. Nazwa skryptu bądź strony, która zostanie wywołana, np. `url: „test.php”`, Kolejnym jest „type”, w naszym przypadku upewnimy się, że będzie to typ domyślny, czyli „GET”, `type: „GET”`, następnie czas na przesłanie danych do skryptu:

```

data: „name=PHPSolution&id=3/2007” ←
      (znany jako $QUERY_STRING)

```

następnie funkcja, która zostanie wywołana, w przypadku wykonania skryptu z sukcesem

```

success: callback_success,

```

Po ukończeniu zapytania, czyli wtedy kiedy spłynęły już wszystkie dane i serwer zamknął połączenie, zostanie wywołana inna funkcja:

```

complete: callback_complete,

```

całość będzie wyglądała zgodnie z listingiem. Wartości zauważenia funkcjami z działu „ajax” są:

- `$.ajaxStart(callback)`;
- `$.ajaxStop(callback)`;
- `$.ajaxSend(callback)`;
- `$.ajaxSuccess(callback)`;
- `$.ajaxComplete(callback)`.

Wszystkie wymienione tu funkcje zwiążą wybraną przez nas funkcję oznaczoną jako „callback” z odpowiednim zdarzeniem. `IAjaxStart` spowoduje, że będzie wykonywana wybrana przez nas funkcja zawsze, kiedy będzie zaczynało wykonywać się jakieś zapytanie „ajax”. `AjaxStop` spowoduje, że wykona się odpowiednia funkcja w chwili zakończenia zapytania typu „ajax”. Funkcja `ajaxSend` wykona napisaną funkcję przed każdym wykonaniem zapytania „ajax”, `ajaxSuccess` odpowiednio po każdym zapytaniu zakończonym sukcesem, a `ajaxComplete` będzie wywoływał funkcję zawsze, kiedy zakończą napływać dane z serwera.

Ponadto funkcja `$.ajax` wspiera zapytania, które jako odpowiedź przesyłają dane w następującej postaci: XML, HTML, script, JSON. Kiedy już poznaliśmy funkcję odpowiedzialną za zapytania asynchroniczne, możemy napisać kod aplikacji webowej, realizującej zaawansowaną walidację formularza. Nikogo nie trzeba przekonywać, że korzystanie

z gotowych funkcji i tego, co już zostało napisane, znacznie przyspiesza tworzenie własnej aplikacji. Z tego właśnie powodu sięgniemy po gotowe funkcje przygotowane do walidacji formularzy, a dostarczone jako plugin jQuery o nazwie: *validation*. Pobieramy plik z adresu: <http://jquery.bassistance.de/validate/jquery.validate.zip> Po rozpakowaniu włączamy w kod naszej strony następujące pliki:

W katalogu „js” umieszczamy następujące pliki pomocnicze: *cmxforms.js*, *jquery.js*, *form.js*, *test.js*, *testrunner.js*. Zaczniemy od przygotowania formularza logowania, który wysłamy asyn-

chronicznie do przetestowania poprawności danych. Kod formularza zawarty jest w Listingu 2.

Kodem zawarty w Listingu 3. oprogramujemy elementy formularza, wykorzystując funkcje dostarczane przez jQuery.

Współpraca jQuery z PHP

Umieszczamy w kodzie HTML potrzebne elementy, takie jak:

```
<button id="reset">Programowy reset formy
    </button>
<div id="result">Oczekiwany rezultat</div>
```

Akcją wykonywaną przy wysłaniu formularza jest asynchroniczne wywołanie skryptu PHP *form.php*, w którym możemy umieścić uwierzytelnienie użytkownika, i włączenie sesji, tak by dalsze wędrowanie po stronie odbywało się w spersonalizowany sposób. Dla potrzeb testów ograniczymy się do prostej symulacji prawdziwego logowania. Kod pliku *form.php* zawiera Listing 4. To, co wyróżnia jQuery od innych podobnych bibliotek, to fakt, że wokół jQuery nabudowało się wiele ciekawych i darmowych bibliotek, które rozszerzają i wzbogacają jego funkcjonalność. Jednym z takich przykładów jest jQPie, który:

- w prosty sposób wykonuje i przetwarza dane ze skryptów PHP, używając metody `$.getJSON`,
- włącza wygenerowany przez PHP kod HTML przy pomocy funkcji `$(element).load`,
- wywołuje funkcje PHP bezpośrednio ze strony przy pomocy funkcji `$.jqpie`,
- wywołuje funkcje jQuery z poziomu PHP w odpowiedzi na wywołania `$.jqpie`.

Innym bardzo dobrym pluginem, integrującym PHP z jQuery, jest „PQuery”. Jest do klasa w PHP, która umożliwia korzystanie na poziomie skryptów php w funkcjonalności jQuery. Aby zainstalować i używać tego modułu, musimy włączyć plik

```
<script src="pq/js/jquery.js" type="text/
    javascript"></script>
```

w kod strony a także w skrypcie php włączyć plik

```
include("pquery/pquery.php");
```

Listing 7. „Hello world” w jQuery

```
<script>
$(document).ready(function() {
    $("a").click(function() {
        alert("Hello world!");
    });
});
</script>
```

Listing 8. Przykład iteracji po elementach strony

```
$("#orderedlist > li").
addClass("blue");
// Kolejnym ułatwieniem jest możliwość
// iteracji po wszystkich tak
// wyselekcjonowanych obiektach.
$("#orderedlist > li").
    each(function(i) {
        $(this).append( " Test " + i );
    });
```

Listing 5. Strona z „jQuery calendar”

```
<HTML><HEAD>
<script type="text/javascript" src="http://code.jquery.com/jquery-latest.pack.js">
</script>
<style type="text/css">@import url(jQuery-calendar.css);</style>
<script type="text/javascript" src="jQuery-calendar.js"></script>
<script type="text/javascript">
$(document).ready(function () {
    $('#calendarFocus').calendar();
    popUpCal.prevText = '<< ';
    popUpCal.nextText = '>> ';
    popUpCal.minDate = new Date(2005, 1 - 1, 1);
    popUpCal.maxDate = new Date(2008, 12 - 1, 31);

});
</script>
</HEAD>
<body>

<form>
<input type="text" class="calendarFocus" id="MyInputData" name="testData1">
</form>

<body>
```

Listing 6. Przykład efektu graficznego w jQuery

```
<HTML><HEAD>
<script type="text/javascript" src="http://code.jquery.com/jquery-latest.pack.js">
</script>
<script type="text/javascript">
$(document).ready(function () {
    $("input").mousedown(function () {
        //alert("");
        $(this).animate({
            opacity: 'hide'
        }, "slow");});
    //$("input").mouseout(function () {
    //    $(this).animate({
    //        opacity: 'show'
    //    }, "slow");});
});
</script>
</HEAD>
<body>
<form>
<input type="text" class="calendarFocus" id="MyInputData" name="testData1">
</form>
<body>
```

Listing 9. Funkcja pomocnicza do dolidacji numeru PESEL

```
function validate(data, jqForm, opts) {
    var form = jqForm[0];
    var pesel = form.pesel.value;
    var vl= validPESEL(pesel);

    if (!vl)
    {

return confirm(„PESEL nie poprawny, czy mimo to wysłać dane?“)
    }

    else return true;
    }
}
```

Listing 10. Zapytanie asynchroniczne get

```
$.get("test.php",
    { name: "PHPSolution", id: "3/2007" },

function(data) {
    $("#wynik").html(data);
}

);
```

Listing 11. Przykładowy plik php, generujący odpowiedź dla zapytania asynchronicznego

```
<?php

If(isset($_GET['name'] ) && $_GET['name'] === „PHPSolution“ ) echo $_GET['name'].
    " To dobry magazyn";

If(isset($_GET['id'] ) && $_GET['id'] === „3/2007“ ) echo $_GET['id'].
    " To ciekawy numer";

?>
```

Listing 12. Przykładowe zapytanie ajax

```
$.ajax(
{
url: „test.php“,
data: „name=PHPSolution&id=3/2007“,
type: „GET“,
success: function(data) {
    $("#wynik").html(data);
},

complete: callback_complete,
});
```

Listing 13. Włączenie potrzebnych plików js dla validatora jQuery

```
<script src="js/jquery.js" type="text/javascript"></script>
<script src="js/cmxfoms.js" type="text/javascript"></script>
<script src="jquery.metadata.js" type="text/javascript"></script>
<script src="jquery.validate.js" type="text/javascript"></script>
```

Po stworzeniu instancji klasy `$pquery = new PQuery();` możemy używać własności jQuery w skrypcie php.

```
<?=$pquery->form_remote_tag( array
('url'=>'index.php?task=ajax',
'update'=>'#idtoupdate'));?>
```

Podaj tekst :

```
<input type="text" name="field" /><br />
<input type="submit" /> </form>
```

Wartość wpisana do pola tekstowego zostanie przekazana zapytaniem asynchronicznym do

skryptu php, a jego odpowiedź zmodyfikuje element `http o id='idtoupdate:'`***

Efekty

Silną stroną biblioteki jQuery są efekty graficzne uzyskiwane za jej pomocą. Mimo że samo jądro dostarcza tylko podstawową i bardzo ubogą funkcjonalność w tej dziedzinie, to jednak narosło bardzo wiele pluginów i dodatków, które możemy włączyć w nasz kod i cieszyć się graficznie i dynamicznie zaawansowanym zachowaniem naszej strony. Przykładem efektywnego wykorzystania możliwości omawianej biblioteki może być wygodny sposób wstawiania daty w wymagane pole tekstowe, za pomocą kalendarza `jQuerycalendar`. Aby zaimplementować tą funkcjonalność, należy pobrać i włączyć w kod HTML oprócz pliku `jquery` dodatkowo `http://marcgrabanski.com/code/jquery-calendar/jquery-calendar.css` oraz `http://marcgrabanski.com/code/jquery-calendar/jquery-calendar.js`. Następnie na wybranym elemencie „input”, wyselekcjonowanym w jQuery, wywołujemy metodę `calendar()`. W zdarzeniu:

```
$('.calendarFocus').calendar();
A w kodzie HTML umieszczamy formularz z
elementem
<input type="text" class="calendarFocus" ←
name="testData">
```

Do pełnego zadowolenia brakuje nam jeszcze konfiguracji kalendarza, tak by domyślna wartość była tą, którą chcemy widzieć w danym miejscu, i format też był odpowiedni.

jQuery ma zaimplementowanych wiele różnych funkcji odpowiedzialnych za wizualne efekty i animacje elementów strony. Prosty przykładem może być tutaj stopniowe nadawanie przezroczystości dowolnemu elementowi strony. Kod w Listingu 6. pokazuje, jak po ustawieniu kursora w polu tekstowym formularza sprawić, by pole to zniknęło.

Podsumowanie

jQuery daje możliwość zarówno budowania szybkich i sprawnych efektów graficznych, jak i łatwego generowania zapytań asynchronicznych i modyfikacji treści strony po stronie przeglądarki.

Przytoczone tu przykłady stanowią cząstkę możliwości jakie daje ta biblioteka. Celem tego artykułu było jednak zapoznanie czytelnika z funkcjonowaniem i sposobem użycia tej biblioteki na własnej stronie i mamy nadzieję, że cel ten został osiągnięty. Mocną stroną biblioteki jest mnogość pluginów i szeroka rzesza programistów gotowa wesprzeć technicznie kiedy mamy jakiś problem, tak więc warto wypróbować to co nam ta biblioteka oferuje.

DARIUSZ DUSZYŃSKI

Autor jest programistą w toruńskiej firmie JADE Sp. z o.o.

Kontakt z autorem: dduszynski@jade.pl