

» Idź do

- Spis treści
- Przykładowy rozdział

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

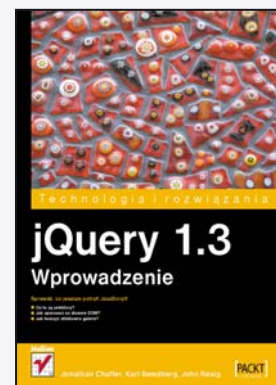
- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2010

jQuery 1.3. Wprowadzenie

Autorzy: Jonathan Chaffer, Karl Swedberg, John Resig
Tłumaczenie: Anna Trojan
ISBN: 978-83-246-2641-0
Tytuł oryginału: Learning jQuery 1.3
Format: 170×230, stron: 424



Język JavaScript po blisko piętnastu latach na rynku dalej ma się dobrze. Interaktywne strony, interesujące efekty oraz technologia AJAX sprawiają, że wciąż jest bardzo atrakcyjnym narzędziem, a aplikacje internetowe z każdym rokiem coraz bardziej zaczynają przypominać te znane z codziennej pracy. Biblioteka jQuery pozwala na jeszcze więcej przy zdecydowanie mniejszym nakładzie pracy! Niemożliwe? Przekonaj się sam!

W trakcie lektury tej książki poznasz tajniki biblioteki jQuery oraz atuty, dzięki którym zyskała ona taką rzeszę fanów. Autorzy omawiają wszystkie zagadnienia związane z tą biblioteką. Na samym początku nauczysz się korzystać z selektorów oraz zdarzeń, aby następnie przejść do zaawansowanych tematów związanych z AJAX-em oraz edycją drzewa DOM. Rozdział poświęcony efektom specjalnym będzie jednym z tych, które pozwolą Ci wykrzesać jeszcze więcej porywających efektów z Twojego kodu. Olśniewające galerie, pokazy slajdów, interaktywne formularze są w zasięgu Twoich możliwości! Pod koniec książki dowiesz się, jak wykorzystywać dodatki oraz tworzyć własne. Trzymasz w rękach długo oczekiwaną na polskim rynku pozycję, poświęconą niezwyklej bibliotece!

- Przeznaczenie biblioteki jQuery
- Wykorzystanie selektorów
- Wykorzystanie mechanizmu zdarzeń
- Zdarzenia złożone
- Użycie efektów specjalnych
- Modyfikowanie arkuszy CSS „w locie”
- Edycja drzewa DOM
- Metody na łatwiejsze wykorzystanie technologii AJAX
- Operacje na tabelach
- Praca z formularzami
- Instalowanie oraz wykorzystanie dodatków
- Tworzenie własnych dodatków dla jQuery

Uzyskaj jeszcze więcej z języka JavaScript!

Spis treści

Przedmowa	11
O autorach	13
O korektorach	15
Wprowadzenie	17
Rozdział 1. Początki	23
Co robi jQuery	23
Dlaczego jQuery działa dobrze	25
Historia projektu jQuery	26
Nasza pierwsza strona z jQuery	27
Pobranie jQuery	27
Utworzenie dokumentu HTML	27
Dodanie jQuery	30
Gotowy produkt	33
Podsumowanie	33
Rozdział 2. Selektory	35
DOM	35
Funkcja fabryczna \$()	36
Selektory CSS	37
Stylizacja poziomów zagnieżdżenia listy	38
Selektory atrybutów	40
Stylizacja odnośników	40
Własne selektory	42
Stylizacja co drugiego wiersza	42
Selektory formularzy	45

Metody przechodzenia drzewa DOM	45
Stylizacja wybranych komórek	46
Łączenie w łańcuch	48
Dostęp do elementów DOM	48
Podsumowanie	49
Rozdział 3. Zdarzenia	51
Wykonywanie zadań w momencie załadowania strony	51
Czas wykonania kodu	51
Wiele skryptów na jednej stronie	52
Skróty poprawiające zwięźłość kodu	53
Współistnienie z innymi bibliotekami	54
Proste zdarzenia	55
Prosty przełącznik stylów	55
Skrótowa obsługa zdarzeń	63
Zdarzenia złożone	64
Pokazywanie i ukrywanie zaawansowanych opcji	64
Wyróżnianie elementów, które można kliknąć	66
Podróż zdarzenia	67
Efekty uboczne propagacji zdarzeń	69
Zmiana podróży — obiekt zdarzenia	70
Element docelowy zdarzenia	71
Zatrzymanie propagacji zdarzeń	71
Działania domyślne	72
Delegacja zdarzeń	72
Usuwanie programu obsługi zdarzeń	75
Przestrzenie nazw zdarzeń	75
Ponowne dowiązanie zdarzenia	76
Symulacja interakcji z użytkownikiem	78
Zdarzenia klawiatury	79
Podsumowanie	81
Rozdział 4. Efekty	83
Modyfikacja CSS w locie	83
Proste ukrywanie i pokazywanie	87
Efekty a szybkość	90
Przyspieszanie	90
Pojawianie się i znikanie	91
Efekty złożone	91
Tworzenie własnych animacji	93
Przełączanie znikania	94
Animacja z użyciem wielu właściwości	94
Efekty jednoczesne a kolejkwane	97
Praca z jednym zbiorem elementów	97
Praca z wieloma zbiorami elementów	100
Funkcje zwrotne	102
W skrócie	104
Podsumowanie	104

Rozdział 5. Edycja drzewa DOM	105
Edycja atrybutów	105
Atrybuty inne od klas	106
Jeszcze raz funkcja fabryczna \$()	108
Wstawianie nowych elementów	110
Przenoszenie elementów	111
Oznaczenie, ponumerowanie i utworzenie odnośnika do kontekstu	115
Dodanie przypisów dolnych	117
Opakowanie elementów	119
Kopiowanie elementów	120
Klonowanie ze zdarzeniami	121
Klonowanie cytatów wyrzuconych	121
Niec o CSS	122
Powrót do kodu	122
Upiększenie cytatów wyrzuconych	125
Metody edycji drzewa DOM w skrócie	126
Podsumowanie	128
Rozdział 6. Ajax	129
Ładowanie danych na żądanie	130
Dodawanie kodu HTML	131
Praca z obiektami JavaScriptu	134
Ładowanie dokumentu XML	141
Wybór formatu danych	144
Przekazywanie danych do serwera	145
Wykonanie żądania GET	146
Wykonanie żądania POST	149
Serializacja formularza	150
Śledzenie żądania	152
Ajax a zdarzenia	155
Ograniczenia w zakresie bezpieczeństwa	156
Wykorzystanie JSONP dla danych zewnętrznych	157
Dodatkowe opcje	158
Niskopoziomowa metoda Ajaksa	159
Modyfikacja opcji domyślnych	159
Ładowanie części strony HTML	160
Podsumowanie	162
Rozdział 7. Przetwarzanie tabel	165
Sortowanie i podział na strony	166
Sortowanie po stronie serwera	166
Sortowanie za pomocą JavaScriptu	167
Paginacja po stronie serwera	183
Paginacja w JavaScriptcie	185
Gotowy kod	190

Modyfikacja wyglądu tabeli	192
Wyróżnianie wierszy	192
Podpowiedzi	199
Zwijanie i rozwijanie części tabeli	205
Filtrowanie	207
Gotowy kod	212
Podsumowanie	215
Rozdział 8. Formularze i ich funkcje	217
Ulepszenie prostego formularza	217
Progresywne ulepszanie stylu formularzy	218
Pola wyświetlane warunkowo	224
Sprawdzanie poprawności formularza	227
Edycja pól wyboru	234
Gotowy kod	237
Zwięzłe formularze	239
Tekst pojemnika na pola formularza	239
Autouzupełnianie oparte na Ajaxie	242
Gotowy kod	250
Praca z danymi liczbowymi formularza	253
Struktura tabeli koszyka z zakupami	253
Odrzucanie danych nieliczbowych	256
Obliczenia arytmetyczne	256
Usuwanie elementów	263
Edycja informacji o wysyłce	267
Gotowy kod	270
Podsumowanie	272
Rozdział 9. Rotacja i przesuwanie elementów	273
Rotacja wiadomości	274
Konfiguracja strony	274
Pobieranie wiadomości z kanału RSS	276
Konfiguracja rotacji elementów	279
Funkcja rotacji wiadomości	280
Pauza po najechnaniu myszą	282
Pobieranie wiadomości RSS z innej domeny	285
Gradientowy efekt blaknięcia	286
Gotowy kod	289
Karuzela obrazków	290
Konfiguracja strony	291
Przesuwanie obrazków po kliknięciu	294
Powiększanie obrazków	301
Gotowy kod	313
Podsumowanie	316

Rozdział 10. Wykorzystywanie dodatków	317
Znalezienie dodatku i uzyskanie pomocy	317
Jak korzystać z dodatku	318
Dodatek Form	319
Wskazówki i sztuczki	320
Biblioteka dodatków jQuery UI	321
Efekty	322
Komponenty interakcji	324
Widżety	326
ThemeRoller w jQuery UI	329
Inne polecane dodatki	330
Formularze	330
Tabele	332
Obrazki	334
Ramki z obrazkami i okna dialogowe	335
Wykresy	338
Zdarzenia	339
Podsumowanie	340
Rozdział 11. Tworzenie dodatków	341
Dodawanie nowych funkcji globalnych	341
Dodanie większej liczby funkcji	342
Jaki to ma sens?	343
Tworzenie nowej metody pomocniczej	344
Dodawanie metod obiektu jQuery	345
Kontekst metody obiektu	346
Łączenie metod w łańcuchy	348
Metody przechodzenia drzewa DOM	349
Dodawanie nowych metod skrótów	353
Parametry metod	356
Proste parametry	358
Tablice asocjacyjne parametrów	359
Domyślne wartości parametrów	360
Funkcje zwrotne	361
Dostosowywanie wartości domyślnych	362
Dodanie wyrażenia selektora	364
Podzielenie się dodatkiem ze światem	367
Konwencje nazewnictwa	367
Użycie aliasu \$	367
Interfejsy metod	368
Styl dokumentacji	368
Podsumowanie	368
Dodatek A Źródła internetowe	369
Dokumentacja jQuery	369
Wiki jQuery	369
jQuery API	369

Przeglądarka jQuery API	370
Visual jQuery	370
Przeglądarka jQuery API w formacie Adobe AIR	370
Informacje o JavaScriptcie	370
Mozilla developer center	370
Dev.opera	370
MSDN JScript Reference	371
Quirksmode	371
JavaScript Toolbox	371
Kompresory kodu w JavaScriptcie	371
YUI Compressor	371
JSMIn	372
Pretty printer	372
Informacje o (X)HTML	372
Strona hipertekstowego języka znaczników W3C	372
Informacje o CSS	372
Strona kaskadowych arkuszy stylów W3C	373
Ściągawka CSS Mezzoblue	373
Position is everything	373
Przydatne blogi	373
Blog jQuery	373
Learning jQuery	374
Ajaxian	374
John Resig	374
JavaScript	374
Robert's talk	374
Web standards with imagination	374
Snook	375
Strona Matta Snidera o JavaScriptcie	375
I can't	375
DOM scripting	375
As days pass by	375
A List Apart	375
Platformy do programowania internetowego korzystające z jQuery	376
Dodatek B Narzędzia programistyczne	377
Narzędzia dla przeglądarki Firefox	377
Firebug	377
Pasek narzędzi Web Developer	378
Venkman	378
Regular Expressions Tester	378
Narzędzia dla przeglądarki Internet Explorer	378
Microsoft Internet Explorer Developer Toolbar	379
Microsoft Visual Web Developer	379
DebugBar	379
Drip	379

Narzędzia dla przeglądarki Safari	380
Programowanie Menu	380
Inspektor www	380
Narzędzia dla przeglądarki Opera	380
Dragonfly	380
Inne narzędzia	381
Firebug Lite	381
NitobiBug	381
Pakiet jQuery dla edytora TextMate	381
Charles	381
Fiddler	382
Aptana	382
Dodatek C Domknięcia w JavaScriptcie	383
Funkcje wewnętrzne	384
Wielka ucieczka	385
Zakresy zmiennych	386
Interakcje między domknięciami	388
Domknięcia w jQuery	389
Argumenty <code>\$(document).ready()</code>	389
Programy obsługi zdarzeń	390
Zagrożenia wynikające z wycieku pamięci	392
Przypadkowe pętle odwołania	393
Problem z wyciekami pamięci w przeglądarce Internet Explorer	394
Dobra wiadomość	395
Podsumowanie	395
Dodatek D Podręczne informacje	397
Wyrażenia selektorów	397
Metody przechodzenia drzewa DOM	399
Metody zdarzeń	400
Metody efektów	403
Metody edycji drzewa DOM	404
Metody Ajaksa	406
Pozostałe metody	407
Skorowidz	409

Początki

Dzisiejszy Internet to dynamiczne środowisko, a jego użytkownicy wysoko stawiają poprzeczkę, zarówno jeśli chodzi o styl, jak i o funkcje stron internetowych. By móc tworzyć interesujące i interaktywne witryny, programiści sięgają po biblioteki JavaScriptu, takie jak jQuery, które pomagają im automatyzować często wykonywane zadania i upraszczać te bardziej skomplikowane. Jednym z powodów, dla których biblioteka jQuery cieszy się takim powodzeniem, jest to, że jest w stanie wspomóc programistę w wielu różnych zadaniach.

Właściwie trudno jest zdecydować, od czego zacząć, ponieważ jQuery ma tyle różnych funkcji. Mimo to struktura biblioteki jest bardzo spójna i symetryczna — większość jej koncepcji zapożyczono z projektów takich, jak **HTML** i **kaskadowe arkusze stylów (CSS)**. Sama struktura biblioteki ułatwia zaczęcie pracy projektantom z niewielkim doświadczeniem programistycznym, ponieważ wielu twórców stron internetowych ma o wiele większe doświadczenie z HTML i CSS niż z JavaScriptem. Już w pierwszym rozdziale książki napiszemy zresztą w jQuery działający program składający się z trzech wierszy kodu. Z drugiej strony również doświadczeni programiści skorzystają na tej konceptualnej spójności, o czym przekonamy się w dalszych, bardziej zaawansowanych rozdziałach.

Przyjrzyjmy się zatem temu, co może dla nas zrobić jQuery.

Co robi jQuery

Biblioteka jQuery udostępnia ogólny poziom abstrakcji służący do tworzenia skryptów na potrzeby często spotykanych zadań, dzięki czemu przydatna jest w niemal każdej sytuacji wymagającej napisania skryptu. Rozszerzalna natura biblioteki oznacza, że nigdy nie udałoby nam się opisać wszystkich jej możliwych zastosowań i funkcji w jednej książce, ponieważ dodatki ciągle są tworzone, dokładając nowe możliwości. Najważniejsze funkcje jądra biblioteki są jednak następujące:

- **Uzyskanie dostępu do elementów dokumentu.** Bez biblioteki JavaScriptu przejście drzewa **DOM** (ang. *Document Object Model*) i lokalizacja wybranych części struktury dokumentu HTML wymaga napisania wielu wierszy kodu. jQuery oferuje rozbudowany i wydajny mechanizm selektorów służący do pobierania fragmentu dokumentu, który ma być przejrany lub zmodyfikowany.
- **Modyfikowanie wyglądu strony internetowej.** CSS oferuje metodę wpływania na sposób wyświetlania dokumentu, jednak na niewiele się przydaje, kiedy nie wszystkie przeglądarki obsługują te same standardy. Dzięki jQuery programiści są w stanie wypełnić tę lukę, polegając na obsłudze tych samych standardów we wszystkich przeglądarkach. Dodatkowo jQuery jest w stanie modyfikować klasy lub pojedyncze właściwości stylu zastosowane do części dokumentu nawet po wygenerowaniu strony.
- **Zmiana zawartości dokumentu.** jQuery nie ogranicza się jednak do zmian czysto kosmetycznych — jest w stanie zmodyfikować zawartość samego dokumentu za pomocą kilku naciśnień klawiatury. Można zmieniać tekst, wstawiać lub podmieniać obrazki, zmieniać kolejność list lub przepisać bądź rozszerzyć całą strukturę HTML — wszystko za pomocą jednego, łatwego w użyciu **API** (ang. *Application Programming Interface*).
- **Reagowanie na interakcję z użytkownikiem.** Nawet najbardziej wyszukane działania nie są szczególnie przydatne, jeśli nie możemy kontrolować tego, kiedy następują. Biblioteka jQuery oferuje elegancki sposób przechwytywania różnego rodzaju zdarzeń — takich jak kliknięcie odnośnika przez użytkownika — bez konieczności zaśmiecania kodu HTML programami obsługi zdarzeń. Jednocześnie API służące do obsługi zdarzeń likwiduje niezgodności między przeglądarkami, które są dla programistów internetowych prawdziwą katorgą.
- **Animacja zmian wprowadzanych do dokumentu.** By efektywnie implementować tego typu działania interaktywne, projektant musi dać użytkownikowi jakieś wizualne informacje zwrotne. Biblioteka jQuery umożliwia to, udostępniając wiele efektów (takich jak blaknięcie czy animowane przejścia), a także służąc jako narzędzie do tworzenia nowych efektów.
- **Pobieranie informacji z serwera bez odświeżania strony.** Ten wzorzec kodu znany jest jako **Ajax** (od *Asynchronous JavaScript and XML*) i wspomaga programistów w tworzeniu bogatych w możliwości oraz reagujących na działanie użytkownika stron. Biblioteka jQuery likwiduje z tego procesu część związaną z różnicami między poszczególnymi przeglądarkami, pozwalając programistom na skupienie się na funkcjonalności po stronie serwera.
- **Uproszczenie często spotykanych zadań wykonywanych za pomocą JavaScriptu.** Oprócz wszystkich możliwości jQuery związanych z obsługą dokumentów biblioteka ta udostępnia także ulepszenia dla podstawowych konstrukcji języka JavaScript, takich jak iteracja i modyfikacja tablic.

Dlaczego jQuery działa dobrze

Wraz ze wzrostem zainteresowania dynamicznym HTML pojawiło się mnóstwo bibliotek i platform opartych na JavaScriptcie. Niektóre z nich są wyspecjalizowane i skupiają się tylko na jednym bądź dwóch z wymienionych wyżej zadań. Inne starają się skatalogować wszystkie możliwe działania i animacje, udostępniając je jako gotowy pakiet. By zachować szeroką gamę możliwości przedstawionych powyżej, ale też niewielki rozmiar, jQuery przyjmuje kilka strategii:

- **Wykorzystanie znajomości CSS.** Opierając mechanizm lokalizacji elementów strony na **selektorach CSS**, jQuery dziedziczy zwięzły, a jednocześnie czytelny sposób wyrażania struktury dokumentu. Biblioteka jQuery staje się punktem wyjścia dla projektantów, którzy chcą dodawać do swych stron różnorodne działania, gdyż znajomość składni CSS jest podstawą zawodowego tworzenia witryn internetowych.
- **Obsługa dodatków.** W celu uniknięcia przeładowania funkcjami jQuery przenosi specjalne zastosowania do **dodatków**. Metoda tworzenia nowych dodatków jest prosta i dobrze udokumentowana, co pobudziło rozwój dużej liczby pomysłów i przydatnych modułów. Nawet w podstawowym pliku jQuery większość opcji wewnętrznie korzysta z architektury dodatków, dzięki czemu można je w miarę potrzeby usunąć, uzyskując jeszcze mniejszą bibliotekę.
- **Abstrakcja niezgodności między przeglądarkami.** Tragedią programistów stron internetowych jest to, że każda przeglądarka ma swój własny zbiór odchyień od opublikowanych standardów. Znaczna część procesu tworzenia każdej aplikacji internetowej przypada na obsługę tych samych funkcji w inny sposób dla każdej z platform. Choć ciągle ewoluujący świat przeglądarek sprawia, że w przypadku bardziej zaawansowanych funkcji stworzenie podstaw kodu całkowicie neutralnych dla przeglądarki jest niemożliwe, jQuery dodaje **poziom abstrakcji** normalizujący często wykonywane zadania, zmniejszający wielkość kodu i znacznie go upraszczający.
- **Praca ze zbiorami.** Kiedy nakazujemy jQuery odnalezienie wszystkich elementów klasy `collapsible`, a następnie ukrycie ich, nie ma potrzeby wykonywania pętli po każdym zwracanym elemencie. Zamiast tego metody takie jak `.hide()` zaprojektowano w taki sposób, by automatycznie działały na zbiorze obiektów, a nie tylko na pojedynczych obiektach. Technika ta, zwana **niejawną iteracją**, oznacza, że wiele konstrukcji pętli staje się zbędnych, co znacznie skraca kod.
- **Zezwolenie na wiele działań w jednym wierszu.** By uniknąć nadmiernego korzystania z tymczasowych zmiennych czy bezsensownego powtarzania, jQuery w większości metod wykorzystuje wzorec programowania zwany **łańcuchem**. Oznacza to, że wynikiem większości działań na obiekcie jest sam obiekt, gotowy do zastosowania na nim kolejnych działań.

Strategie te pozwalają na utrzymanie niewielkiego rozmiaru pakietu jQuery — po skompresowaniu jest to poniżej 20 KB — udostępniając jednocześnie techniki umożliwiające zachowanie zwięzłości własnego kodu korzystającego z biblioteki.

Elegancja biblioteki po części jest cechą samego projektu jQuery, a po części efektem procesu ewolucyjnego pobudzanego przez aktywną społeczność, która wytworzyła się wokół projektu. Użytkownicy jQuery aktywnie omawiają nie tylko rozwój dodatków, ale także ulepszenia samego jądra biblioteki. W dodatku A przedstawimy wiele zasobów dostępnych dla programistów korzystających z jQuery.

Pomimo skali wysiłków niezbędnych do stworzenia tak elastycznego i rozbudowanego systemu produkt końcowy jest darmowy i dostępny dla każdego. Projekt ten dostępny jest na podwójnej licencji: **GNU Public License** (dzięki czemu można go dołączyć do wielu innych projektów open source) oraz **MIT License** (by ułatwić zastosowanie jQuery w oprogramowaniu własnościowym).

Historia projektu jQuery

Niniejsza książka omawia funkcjonalność i składnię **jQuery 1.3.x**, najnowszą wersję dostępną w czasie jej pisania. Założenia leżące u podstaw biblioteki — udostępnienie łatwych sposobów odnajdywania elementów na stronie internetowej oraz modyfikowania ich — nie zmieniły się w trakcie jej rozwijania, choć zmienione zostały szczegóły składni i dostępnych możliwości. Poniższy krótki przegląd historii projektu opisuje najważniejsze zmiany pomiędzy wersjami.

- **Publiczne ogłoszenie prac:** John Resig po raz pierwszy wspominał o poprawkach do biblioteki Prototype (części „Behaviour”) w sierpniu 2005 roku. Nowa biblioteka została oficjalnie opublikowana pod nazwą **jQuery** 14 stycznia 2006 roku.
- **jQuery 1.0 (sierpień 2006):** już pierwsze wydanie biblioteki miało rozbudowaną obsługę selektorów CSS, zdarzeń i interakcji opartych na Ajaxie.
- **jQuery 1.1 (styczeń 2007):** to wydanie znacznie usprawniło API. Wiele rzadko wykorzystywanych metod zostało połączonych, co zmniejszyło całkowitą liczbę metod, które trzeba opanować i udokumentować.
- **jQuery 1.1.3 (lipiec 2007):** to pomniejsze wydanie zawierało ogromną poprawę szybkości działania silnika selektorów jQuery. Od tej wersji wydajność jQuery wypada bardzo korzystnie na tle podobnych bibliotek JavaScriptu, takich jak Prototype, Mootools oraz Dojo.
- **jQuery 1.2 (wrzesień 2007):** w wydaniu tym usunięto składnię **XPath** służącą do wybierania elementów, gdyż była ona powtarzalna w stosunku do składni CSS. Dostosowywanie efektów do własnych potrzeb jest od tego wydania bardziej elastyczne; tworzenie dodatków stało się łatwiejsze dzięki wprowadzeniu **zdarzeń z przestrzeni nazw**.
- **jQuery UI (wrzesień 2007):** ogłoszono powstanie nowego zbioru dodatków, który miał zastąpić popularny, jednak starzejący się dodatek Interface. Obejmował on bogatą kolekcję gotowych widżetów, a także zbiór narzędzi służących do budowania bardziej zaawansowanych elementów, takich jak interfejsy oparte na zasadzie „przeciągnij i upuść”.

- **jQuery 1.2.6 (maj 2008)**: funkcjonalność popularnego dodatku Brandona Aarona o nazwie **Dimensions** została dołączona do jądra biblioteki.
- **jQuery 1.3 (styczeń 2009)**: poważna przebudowa silnika selektorów (**Sizzle**) ogromnie poprawiła wydajność biblioteki. Oficjalnie obsługiwana jest również delegacja zdarzeń.

Informacje o starszych wersjach jQuery można znaleźć na stronie internetowej projektu pod adresem: http://docs.jquery.com/History_of_jQuery.

Nasza pierwsza strona z jQuery

Skoro omówiliśmy już możliwości udostępniane przez jQuery, możemy teraz sprawdzić, jak można wykorzystać tę bibliotekę w praktyce.

Pobranie jQuery

Oficjalna strona internetowa jQuery (<http://jquery.com/>) jest zawsze najlepszym adresem, pod którym można znaleźć aktualny kod i informacje dotyczące biblioteki. Na początek potrzebna jest nam kopia jQuery, którą można pobrać bezpośrednio na stronie głównej tej witryny. W każdym momencie dostępnych może być kilka wersji jQuery — dla nas jako programistów stron internetowych najbardziej odpowiednia będzie najnowsza nieskompresowana wersja biblioteki. W środowisku produkcyjnym można ją zastąpić wersją skompresowaną.

Nie jest wymagana żadna instalacja. By użyć jQuery, wystarczy umieścić plik na naszej stronie internetowej, w publicznie dostępnym miejscu. Ponieważ JavaScript jest językiem interpretowanym, nie musimy martwić się o fazę kompilacji. Zawsze gdy będziemy potrzebować jQuery na stronie internetowej, będziemy się po prostu odnosić do lokalizacji pliku z dokumentu HTML.

Utworzenie dokumentu HTML

Większość przykładów wykorzystujących jQuery składa się z trzech elementów: samego dokumentu HTML, plików CSS nadających mu styl i plików JavaScriptu pozwalających wykonywać działania. W naszym pierwszym przykładzie wykorzystamy stronę z fragmentem książki¹; do części kodu przypisano kilka klas.

¹ W przykładzie wykorzystano fragment książki *Po drugiej stronie lustra* Lewisa Carrolla w tłumaczeniu Roberta Stillera — *przyp. tłum.*

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl" lang="pl">

<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Po drugiej stronie lustra</title>
  <link rel="stylesheet" href="alice.css" type="text/css" media="screen" />
  <script src="jquery.js" type="text/javascript"></script>
  <script src="alice.js" type="text/javascript"></script>
</head>

<body>
  <h1>Po drugiej stronie lustra</h1>
  <div class="author">autor: Lewis Carroll</div>
  <div class="chapter" id="chapter-1">
    <h2 class="chapter-title">1. Dom odbity w lustrze</h2>
    <p>Na stole koło Alicji leżała książka, więc kiedy tak siedziała,
    obserwując Białego Króla (bo wciąż trochę się o niego lękała i trzymała
    w pogotowiu atrament, żeby go oblać, gdyby jeszcze raz zemdleła),
    przewracała sobie kartki w poszukiwaniu czegoś, co by mogła przeczytać:
    <span class="spoken">"bo wszystko to w jakimś nieznanym języku!"</span>
    powiedziała do siebie.</p>
    <p>Brzmiało to tak.</p>
    <div class="poem">
      <h3 class="poem-title">IKAŁORBAŻ</h3>
      <div class="poem-stanza">
        <div>ywtårks eikbilš ,ywałtzsurm szac łyB</div>
        <div>,yłpyriwš cączreiw hcażław aN</div>
        <div>ywtąłgorob anc od ełgzim A</div>
        <div>.yłpyzrgz yruczsiwš enmodz I</div>
      </div>
    </div>
    <p>Zastanawiała się nad tym przez jakiś czas, aż błysnęła jej genialna
    myśl. <span class="spoken">"Przecież to jest, oczywiście, Lustrzana
    Książka! i jak na nią popatrzę w Lustrze, to słowa znów będą takie,
    jak trzeba."</span></p>
    <p>I odczytała następujący wiersz:</p>
    <div class="poem">
      <h3 class="poem-title">ŻABROŁAKI</h3>
      <div class="poem-stanza">
        <div>Był czas mrusztławy, ślibkie skrątwy</div>
        <div>Na wałzach wiercząc świrypty,</div>
        <div>A mizgłe do cna borogławy</div>
        <div>I zdomne świszczury zgrzypły.</div>
      </div>
    </div>
  </div>
</body>
</html>

```

Układ plików na serwerze nie ma znaczenia. Odwołania z jednego pliku do drugiego należy jedynie dostosować tak, by pasowały do wybranego przez nas sposobu organizacji. W większości przykładów w książce w odwołaniach do plików wykorzystamy ścieżki względne (*../images/foo.png*), a nie bezwzględne (*/images/foo.png*). Pozwala to na lokalne uruchamianie kodu bez konieczności posiadania serwera WWW.

Natychmiast po zwykłym nagłówku HTML ładowany jest arkusz stylów. W tym przykładzie jest on wyjątkowo spartański.

```
body {
    font: 62.5% Arial, Verdana, sans-serif;
}
h1 {
    font-size: 2.5em;
    margin-bottom: 0;
}
h2 {
    font-size: 1.3em;
    margin-bottom: .5em;
}
h3 {
    font-size: 1.1em;
    margin-bottom: 0;
}
.poem {
    margin: 0 2em;
}
.highlight {
    font-style: italic;
    border: 1px solid #888;
    padding: 0.5em;
    margin: 0.5em 0;
    background-color: #ffc;
}
```

Po odwołaniu do arkusza stylów dołączone zostają pliki JavaScriptu. Istotne jest, by element `script` z biblioteką jQuery znajdował się *przed* elementem `script` dla naszych własnych skryptów. W przeciwnym razie biblioteka jQuery nie będzie dostępna, kiedy nasz kod będzie się próbował do niej odwołać.

W pozostałej części książki drukowane będą jedynie istotne części plików HTML i CSS. Pełne pliki dostępne są na stronie internetowej książki pod adresem: <http://helion.pl/ksiazki/jquery1.htm>.

Teraz nasza strona wygląda tak:

Po drugiej stronie lustra

autor: Lewis Carroll

1. Dom odbity w lustrze

Na stole koło Alicji leżała książka, więc kiedy tak siedziała, obserwując Białego Króla (bo wciąż trochę się o niego lękała i trzymała w pogotowiu atrament, żeby go oblać, gdyby jeszcze raz zemdlął), przewracała sobie karty w poszukiwaniu czegoś, co by mogła przeczytać: "bo wszystko to w jakimś nieznanym języku!" powiedziała do siebie.

Brzmiało to tak:

IKAŁORBAŻ

ywtąks eikbilś ,ywaltzsum szoc tyB
 .ytpyriwś cązoreiw hcazław aN
 ywtąlgorob anc od ełgzim A
 .ytpyzrgz yruczsiwś enmodz I

Zastanawiała się nad tym przez jakiś czas, aż błysnęła jej genialna myśl. "Przecież to jest, oczywiście, Lustrzana Książka! i jak na nią popatrzę w Lustrze, to słowa znów będą takie, jak trzeba."

I odczytała następujący wiersz:

ŻABROŁAKI

Był czas mrusztławy, śliskie skrątlwy
 Na wałkach wierząc świrypły,
 A mizgłe do ona borogławy
 I zdomne świszczury zgrzypty.

Rysunek 1.1.

Wykorzystamy teraz jQuery do nadania nowego stylu tekstowi wiersza.

Przykład ten ma na celu zademonstrowanie prostego zastosowania jQuery. W rzeczywistości ten typ stylizacji tekstu można wykonać za pomocą samego CSS.

Dodanie jQuery

Nasz kod umieścimy w drugim — obecnie pustym — pliku JavaScriptu, dołączonym do dokumentu HTML za pomocą kodu `<script src="alice.js" type="text/javascript"></script>`. W tym przykładzie wystarczy nam trzy wiersze kodu:

```
$(document).ready(function() {
    $('.poem-stanza').addClass('highlight');
});
```

Odnalezienie tekstu wiersza

Podstawowym działaniem jQuery jest wybranie części dokumentu. Wykonywane jest to za pomocą konstrukcji `$()`. Zazwyczaj przyjmuje ona jako parametr łańcuch znaków, który może zawierać dowolne wyrażenie selektora CSS. W tym przypadku chcemy odnaleźć wszystkie części dokumentu, do których przypisano klasę `poem-stanza`, dlatego selektor jest bardzo prosty. W książce opiszemy jednak również o wiele bardziej wyszukane możliwości. Różne sposoby odnajdywania części dokumentu omówimy w rozdziale 2.

Funkcja `$()` jest tak naprawdę fabryką dla obiektu **jQuery** będącego podstawowym budulcem, z jakim będziemy od teraz pracować. Obiekt **jQuery** zawiera zero lub większą liczbę elementów drzewa DOM i pozwala nam wchodzić z nimi w różnego rodzaju interakcje. W tym przypadku chcemy zmodyfikować wygląd części strony, co osiągniemy, zmieniając klasy przypisane do tekstu wiersza.

Wstawienie nowej klasy

Metoda `.addClass()`, tak jak większość metod **jQuery**, ma opisową nazwę (ang. *add class* — „dodaj klasę”). Dodaje ona klasę CSS do wybranej części strony. Jej jedynym parametrem jest nazwa klasy, jaką należy dodać. Metoda ta oraz jej odpowiednik `.removeClass()` pozwalają nam łatwo zaobserwować działanie **jQuery**, kiedy będziemy badać różne dostępne wyrażenia selektorów. Na razie nasz przykład po prostu dodaje klasę `highlight`, którą nasz arkusz stylów definiuje jako tekst z obramowaniem napisany kursywą.

Warto zauważyć, że w celu dodania klasy do wszystkich zwrotek wiersza (elementów poem-stanza) nie jest konieczne wykonywanie jakiegokolwiek iteracji. Tak jak wspominaliśmy, **jQuery** wykorzystuje w metodach takich jak `.addClass()` **iterację niejawną**, dzięki czemu jedno wywołanie funkcji wystarczy do zmodyfikowania wszystkich wybranych części dokumentu.

Wykonanie kodu

Połączenie `$()` i `.addClass()` wystarczy nam do osiągnięcia celu, jakim jest zmiana wyglądu tekstu wiersza. Jeśli jednak ten wiersz kodu wstawimy po prostu w nagłówku dokumentu, nie przyniesie to żadnego efektu. Kod w JavaScriptcie wykonywany jest w momencie napotkania go przez przeglądarkę, a w czasie gdy przetwarzany jest nagłówek, nie ma jeszcze żadnego kodu HTML, któremu można by nadać styl. Musimy zatem opóźnić wykonanie kodu do czasu, gdy będzie dla nas dostępne drzewo DOM.

Tradycyjnym mechanizmem opóźniania wykonywania kodu w JavaScriptcie jest wywołanie kodu wewnątrz **programu obsługi zdarzeń** (ang. *event handler*). Większość programów obsługi zdarzeń dostępna jest dla zdarzeń inicjowanych przez użytkownika, takich jak kliknięcia myszą i naciśnięcia klawiszy. Gdybyśmy nie mieli dostępu do **jQuery**, musielibyśmy polegać na programie obsługi zdarzeń `onload`, który wywoływany jest po wygenerowaniu strony (ze wszystkimi obrazkami). By wywołać kod ze zdarzenia `onload`, umieścilibyśmy go wewnątrz funkcji:

```
function highlightPoemStanzas() {
    $(''.poem-stanza').addClass('highlight');
}
```

Następnie dołączylibyśmy tę funkcję do zdarzenia, modyfikując element `<body>` dokumentu HTML, tak by się do niej odwoływał:

```
<body onload="highlightPoemStanzas();">
```

W ten sposób nasz kod zostałby wykonany po całkowitym załadowaniu strony.

To rozwiązanie ma jednak swoje wady. Zmodyfikowaliśmy sam kod HTML, by wywołać zmianę zachowania. Tak ściśle powiązanie struktury i funkcji zaśmieca kod, wymagając nieraz powtarzania wywołań tej samej funkcji na wielu różnych stronach lub — w przypadku zdarzeń takich jak kliknięcie myszą — przy każdym pojawieniu się elementu na stronie. Dodanie nowych działań wymaga wtedy wprowadzania zmian w wielu miejscach, co zwiększa szansę na popełnienie błędu i komplikuje równoległą pracę nad wieloma dokumentami projektantom i programistom.

By uniknąć tych problemów, jQuery pozwala zaplanować wywołanie funkcji na czas po załadowaniu DOM — bez czekania na obrazki — dzięki konstrukcji `$(document).ready()`. Z funkcją zdefiniowaną jak powyżej możemy zapisać:

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

Technika ta nie wymaga wprowadzania jakichkolwiek modyfikacji do kodu HTML. Zamiast tego działanie jest dołączane w całości za pomocą pliku JavaScriptu. W rozdziale 3. dowiemy się, jak można odpowiadać na inne typy zdarzeń, rozdzielając ich efekty od struktury dokumentu HTML.

To wcielenie kodu jest jeszcze trochę nieoptymalne, ponieważ funkcja `highlightPoemStanzas()` definiowana jest tylko po to, by skorzystać z niej natychmiast — i to tylko raz. Oznacza to, że wykorzystaliśmy identyfikator w globalnej przestrzeni nazw funkcji, zyskując niewiele, a teraz musimy pamiętać, by z niego już więcej nie korzystać. JavaScript, podobnie jak inne języki programowania, zawiera sposób obejścia tego braku optymalności zwany **funkcjami anonimowymi** (czasami nazywany również **funkcjami lambda**). Za pomocą funkcji anonimowych możemy pisać kod w sposób, który został zaprezentowany na początku:

```
$(document).ready(function() {
    $('.poem-stanza').addClass('highlight');
});
```

Wykorzystując słowo kluczowe `function` bez nazwy funkcji, definiujemy funkcję dokładnie tam, gdzie jest potrzebna, a nie wcześniej. Usuwa to niepotrzebne rozbudowanie kodu i pozwala zredukować go do trzech wierszy JavaScriptu. Skrót ten jest szczególnie wygodny w przypadku kodu jQuery, gdyż wiele metod przyjmuje jako argument funkcję, a funkcje takie rzadko można wykorzystać ponownie.

Kiedy składnię tę wykorzystujemy do zdefiniowania funkcji anonimowej wewnątrz ciała innej funkcji, można utworzyć **domknięcie** (ang. *closure*). Jest to bardziej zaawansowana koncepcja o szerokich możliwościach, jednak powinno się ją dobrze zrozumieć, gdy intensywnie wykorzystuje się definicje zagnieżdżonych funkcji, gdyż może ona mieć niezamierzone konsekwencje i implikacje w zakresie wykorzystania pamięci. Kwestię tę omawiamy w całości w dodatku C.

Gotowy produkt

Kiedy nasz kod w JavaScriptcie jest gotowy, strona wygląda tak:

Po drugiej stronie lustra

autor: Lewis Carroll

1. Dom odbity w lustrze

Na stole koło Alicji leżała książka, więc kiedy tak siedziała, obserwując Białego Króla (bo wciąż trochę się o niego lękała i trzymała w pogotowiu atrament, żeby go oblać, gdyby jeszcze raz zemdlął), przewracała sobie kartki w poszukiwaniu czegoś, co by mogła przeczytać: "bo wszystko to w jakimś nieznanym języku!" powiedziała do siebie.

Brzmiało to tak:

IKAŁORBAŻ

*ywfąks eikbilś ,ywłtzsum sazo tyB
 ,ypyrwiś czązoreiw hcazlaw aN
 ywfąlgorob ano od ełgżim A
 ,ypyzrgz yruzcziwiś enmodz I*

Zastanawiała się nad tym przez jakiś czas, aż blysnęła jej genialna myśl. "Przecież to jest, oczywiście, Lustrzana Książka! i jak na nią popatrzę w Lustrze, to słowa znów będą takie, jak trzeba."

I odczytała następujący wiersz:

ŻABROŁAKI

*Był czas mruszławy, ślibkie skrańwy
 Na wążach wierząc świrypty,
 A mizgle do ona borogławy
 I zdonne świszczury zgrzypty.*

Rysunek 1.2.

Zwrotki wierszy napisane są teraz kursywą i umieszczone są w ramce, zgodnie z arkuszem stylów *alice.css*, z powodu wstawienia klasy `highlight` za pomocą kodu w JavaScriptcie.

Podsumowanie

Wiemy już, dlaczego programista mógłby wybrać korzystanie z biblioteki JavaScriptu, zamiast pisać cały kod od podstaw — nawet w przypadku najprostszych zadań. Widzieliśmy także, co sprawia, że jQuery sprawdza się jako doskonała biblioteka, której zastosowanie może być lepszym rozwiązaniem od innych opcji. Wiemy także mniej więcej, jakiego typu zadania jQuery może uprościć.

W niniejszym rozdziale nauczyliśmy się, jak udostępnić jQuery kodowi w JavaScriptcie na naszej stronie internetowej, jak wykorzystać funkcję fabryczną `$()` do zlokalizowania części strony z przypisaną określoną klasą, jak wywoływać metodę `.addClass()` w celu nadania tej części strony dodatkowego stylu, a także jak wywołać `$(document).ready()`, co spowoduje, że kod ten zostanie wywołany po załadowaniu strony.

Prosty przykład, którym się posłużyliśmy, demonstruje sposób działania jQuery, jednak nie jest szczególnie przydatny w praktyce. W kolejnym rozdziale rozbudujemy ten kod, zapoznając się z wyszukaniem językiem selektorów jQuery i znajdując praktyczne zastosowania dla tej techniki.