

Bear Bibeault, Yehuda Katz, Aurelio De Rosa

jQuery *w akcji*



WYDANIE III

Helion 

Tytuł oryginału: jQuery in Action, Third Edition

Tłumaczenie: Piotr Pilch

Projekt okładki: Studio Gravite / Olsztyn

Obarek, Pokoński, Pazdrijowski, Zaprucki

ISBN: 978-83-283-2275-2

Original edition copyright © 2015 by Manning Publications Co.

All rights reserved.

Polish edition copyright © 2016 by HELION SA.

All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/jquery3.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/jquery3>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

<i>Słowo wstępne do wydania trzeciego</i>	15
<i>Słowo wstępne do wydania pierwszego</i>	17
<i>Przedmowa</i>	19
<i>Podziękowania</i>	21
<i>O książce</i>	23
<i>O autorach</i>	27

CZĘŚĆ I BIBLIOTEKA JQUERY — PIERWSZE KROKI 29

Rozdział 1. Wprowadzenie do biblioteki jQuery 31

- 1.1. Pisz mniej, a rób więcej 32
- 1.2. Przejrzysty kod JavaScript 34
 - 1.2.1. Oddzielanie zachowania od struktury 35
 - 1.2.2. Oddzielanie skryptu 36
- 1.3. Instalowanie biblioteki jQuery 37
 - 1.3.1. Wybieranie właściwej wersji 37
 - 1.3.2. Zwiększanie wydajności przy użyciu sieci CDN 40
- 1.4. Struktura biblioteki jQuery 42
 - 1.4.1. Tworzenie własnej wersji niestandardowej zmniejszającej wielkość pliku 43
- 1.5. Fundamenty biblioteki jQuery 44
 - 1.5.1. Właściwości, narzędzia i metody 44
 - 1.5.2. Obiekt biblioteki jQuery 45
 - 1.5.3. Procedura obsługi zdarzenia gotowości dokumentu 47
- 1.6. Podsumowanie 49

CZĘŚĆ II PODSTAWOWE SKŁADNIKI BIBLIOTEKI JQUERY 51

Rozdział 2. Wybieranie elementów 53

- 2.1. Wybieranie elementów do modyfikacji 54
- 2.2. Podstawowe selektory 56
 - 2.2.1. Selektor „wszystko” (lub uniwersalny) 57
 - 2.2.2. Selektor identyfikatora 60
 - 2.2.3. Selektor klasy 61
 - 2.2.4. Selektor elementu 61
- 2.3. Pobieranie elementów na podstawie ich hierarchii 62
- 2.4. Wybieranie elementów za pomocą atrybutów 64

- 2.5. Wprowadzenie do filtrów 68
 - 2.5.1. Filtry położenia 68
 - 2.5.2. Filtry elementów podrzędnych 69
 - 2.5.3. Filtry formularza 73
 - 2.5.4. Filtry treści 74
 - 2.5.5. Inne filtry 75
 - 2.5.6. Tworzenie filtrów niestandardowych 77
- 2.6. Zwiększanie wydajności za pomocą kontekstu 80
- 2.7. Sprawdzanie umiejętności przy użyciu ćwiczeń 81
 - 2.7.1. Ćwiczenia 82
 - 2.7.2. Rozwiązania 82
- 2.8. Podsumowanie 83

Rozdział 3. Przetwarzanie kolekcji biblioteki jQuery 85

- 3.1. Generowanie nowych elementów HTML 86
- 3.2. Zarządzanie kolekcją biblioteki jQuery 89
 - 3.2.1. Określanie wielkości zestawu 91
 - 3.2.2. Uzyskiwanie elementów zestawu 91
 - 3.2.3. Uzyskiwanie zestawów za pomocą relacji 96
 - 3.2.4. Dopasowywanie zestawu 101
 - 3.2.5. Jeszcze więcej sposobów użycia zestawu 110
- 3.3. Podsumowanie 113

Rozdział 4. Użycie właściwości, atrybutów i danych 115

- 4.1. Definiowanie właściwości i atrybutów elementów 116
- 4.2. Użycie atrybutów 119
 - 4.2.1. Pobieranie wartości atrybutów 119
 - 4.2.2. Ustawianie wartości atrybutów 120
 - 4.2.3. Usuwanie atrybutów 122
 - 4.2.4. Zabawa z atrybutami 123
- 4.3. Modyfikowanie właściwości elementu 125
- 4.4. Przechowywanie danych niestandardowych w elementach 128
- 4.5. Podsumowanie 135

Rozdział 5. Ożywianie stron za pomocą biblioteki jQuery 137

- 5.1. Zmienianie stylów elementów 138
 - 5.1.1. Dodawanie i usuwanie nazw klas 138
 - 5.1.2. Uzyskiwanie i ustawianie stylów 143
- 5.2. Określanie zawartości elementu 153
 - 5.2.1. Zastępowanie kodu HTML lub treści tekstowej 153
 - 5.2.2. Przenoszenie elementów 155
 - 5.2.3. Opakowywanie elementów i usuwanie ich opakowania 161
 - 5.2.4. Usuwanie elementów 165

5.2.5.	Powielanie elementów	167
5.2.6.	Zastępowanie elementów	168
5.3.	Obsługa wartości elementów formularza	170
5.4.	Podsumowanie	172
Rozdział 6. Zdarzenia są tam, gdzie coś się dzieje		173
6.1.	Modele zdarzeń przeglądark	174
6.1.1.	Model zdarzeń poziomu DOM Level 0	175
6.1.2.	Model zdarzeń poziomu DOM Level 2	183
6.1.3.	Model przeglądarki Internet Explorer	189
6.2.	Model zdarzeń biblioteki jQuery	189
6.2.1.	Dołączanie procedur obsługi zdarzeń w bibliotece jQuery	190
6.2.2.	Usuwanie procedur obsługi zdarzeń	198
6.2.3.	Inspekcja instancji obiektu Event	200
6.2.4.	Wyzwalanie procedur obsługi zdarzeń	201
6.2.5.	Metody skrócone	206
6.2.6.	Sposób tworzenia zdarzeń niestandardowych	210
6.2.7.	Określanie przestrzeni nazw dla zdarzeń	211
6.3.	Podsumowanie	212
Rozdział 7. Demonstracja lokalizatora dysków DVD		215
7.1.	Praktyczne zastosowanie zdarzeń (i nie tylko)	216
7.1.1.	Filtrowanie dużych zestawów danych	217
7.1.2.	Tworzenie elementów za pomocą replikacji szablonu	219
7.1.3.	Tworzenie podstawowego kodu znaczników	221
7.1.4.	Dodawanie nowych filtrów	222
7.1.5.	Dodawanie szablonów kontrolek	225
7.1.6.	Usuwanie niepożądanych filtrów oraz inne zadania	227
7.1.7.	Wyświetlanie wyników	227
7.1.8.	Zawsze można jeszcze coś ulepszyć	229
7.2.	Podsumowanie	231
Rozdział 8. Wzbogacanie stron przy użyciu animacji i efektów		233
8.1.	Wyświetlanie i ukrywanie elementów	234
8.1.1.	Implementowanie „modułu” umożliwiającego zwijanie	235
8.1.2.	Przełączanie stanu wyświetlania elementów	238
8.2.	Animowanie stanu wyświetlania elementów	238
8.2.1.	Stopniowe wyświetlanie i ukrywanie elementów	239
8.2.2.	Wprowadzenie do strony laboratorium efektów biblioteki jQuery	243
8.2.3.	Stopniowe rozjaśnianie i ściemnianie elementów	245
8.2.4.	Przesuwanie elementów w górę i w dół	248
8.2.5.	Zatrzymywanie animacji	249
8.3.	Dodawanie kolejnych funkcji sposobu animacji do biblioteki jQuery	251

- 8.4. Tworzenie animacji niestandardowych 253
 - 8.4.1. Niestandardowa animacja skalowania 256
 - 8.4.2. Niestandardowa animacja efektu opadania 257
 - 8.4.3. Niestandardowa animacja efektu rozpraszania 258
- 8.5. Animacje i kolejkowanie 260
 - 8.5.1. Jednocześnie wykonywane animacje 260
 - 8.5.2. Kolejkowanie funkcji do wykonania 262
 - 8.5.3. Wstawianie funkcji do kolejek efektów 268
- 8.6. Podsumowanie 269

Rozdział 9. Poza modelem DOM — funkcje narzędziowe biblioteki jQuery 271

- 9.1. Użycie właściwości biblioteki jQuery 272
 - 9.1.1. Wylęczanie animacji 273
 - 9.1.2. Zmiana szybkości animacji 273
 - 9.1.3. Właściwość \$.support 274
- 9.2. Użycie innych bibliotek z biblioteką jQuery 275
- 9.3. Modyfikowanie obiektów i kolekcji języka JavaScript 279
 - 9.3.1. Obcinanie łańcuchów 280
 - 9.3.2. Iteracja właściwości i kolekcji 281
 - 9.3.3. Filtrowanie tablic 283
 - 9.3.4. Translacja tablic 285
 - 9.3.5. Więcej zabawy z tablicami języka JavaScript 286
 - 9.3.6. Rozszerzanie obiektów 290
 - 9.3.7. Serializacja wartości parametrów 292
 - 9.3.8. Testowanie obiektów 296
 - 9.3.9. Analizowanie funkcji 299
- 9.4. Różne funkcje narzędziowe 302
 - 9.4.1. Nie wykonując żadnego działania 302
 - 9.4.2. Sprawdzanie przynależności 303
 - 9.4.3. Wstępne wiązanie kontekstów funkcji 304
 - 9.4.4. Określanie wartości wyrażeń 306
 - 9.4.5. Zgłaszanie wyjątków 307
- 9.5. Podsumowanie 308

Rozdział 10. Komunikacja z serwerem z wykorzystaniem technologii Ajax 309

- 10.1. Odświeżenie informacji o technologii Ajax 310
 - 10.1.1. Tworzenie instancji obiektu XHR 310
 - 10.1.2. Inicjowanie żądania 313
 - 10.1.3. Śledzenie postępu działań 314
 - 10.1.4. Uzyskiwanie odpowiedzi 314
- 10.2. Ładowanie zawartości do elementów 315
 - 10.2.1. Ładowanie zawartości za pomocą kodu jQuery 317
 - 10.2.2. Ładowanie fragmentów dynamicznego kodu HTML 321

10.3.	Tworzenie żądań GET i POST	325
10.3.1.	Uzyskiwanie danych za pomocą metody GET	327
10.3.2.	Uzyskiwanie danych formatu JSON	329
10.3.3.	Dynamiczne ładowanie skryptu	330
10.3.4.	Tworzenie żądań metody POST	332
10.3.5.	Implementowanie kaskadowych elementów rozwijanych	333
10.4.	Przejmowanie pełnej kontroli nad żądaniem Ajax	338
10.4.1.	Tworzenie żądań Ajax z wszystkimi dodatkami	338
10.4.2.	Ustawianie wartości domyślnych żądania	342
10.4.3.	Obsługa zdarzeń Ajax	344
10.4.4.	Zaawansowane funkcje narzędziowe Ajax	348
10.5.	Podsumowanie	350

Rozdział 11. Demonstracja formularza kontaktowego wykorzystującego technologię Ajax 351

11.1.	Funkcje projektu	352
11.2.	Tworzenie kodu znaczników	354
11.3.	Implementowanie kodu serwerowego PHP	355
11.4.	Sprawdzanie poprawności pól z wykorzystaniem technologii Ajax	357
11.5.	Jeszcze więcej zabawy z technologią Ajax	359
11.5.1.	Ukrywanie okna dialogowego	361
11.6.	Poprawianie komfortu pracy użytkownika za pomocą efektów	362
11.6.1.	Przełączanie efektów	362
11.7.	Uwaga dotycząca dostępności	363
11.8.	Podsumowanie	365

CZĘŚĆ III ZAAWANSOWANE ZAGADNIENIA 367

Rozdział 12. Gdy biblioteka jQuery nie wystarcza, na ratunek przychodzą dodatki 369

12.1.	Dlaczego warto rozszerzać bibliotekę jQuery?	370
12.2.	Znajdowanie dodatków	371
12.2.1.	Metoda użycia (dobrze zaprojektowanego) dodatku	371
12.2.2.	Znakomite dodatki do zastosowania w projektach	375
12.3.	Wytyczne dotyczące tworzenia dodatku biblioteki jQuery	375
12.3.1.	Konwencje nazewnictwa dotyczące plików i funkcji	376
12.3.2.	Uważaj na skrót \$	378
12.3.3.	Ujarzmianie złożonych list parametrów	378
12.3.4.	Pozostań przy jednej przestrzeni nazw	381
12.3.5.	Określanie przestrzeni nazw dla zdarzeń i danych	384
12.3.6.	Utrzymywanie możliwości tworzenia łańcucha metod	389
12.3.7.	Zapewnienie dostępu publicznego do ustawień domyślnych	389

- 12.4. Demonstracja tworzenia pokazu slajdów w postaci dodatku biblioteki jQuery 392
 - 12.4.1. Definiowanie kodu znaczników 394
 - 12.4.2. Tworzenie dodatku Jqia Photomatic 396
- 12.5. Tworzenie niestandardowych funkcji narzędziowych 403
 - 12.5.1. Tworzenie funkcji formatującej datę 404
- 12.6. Podsumowanie 408

Rozdział 13. Uniknięcie „piekła wywołań zwrotnych” za pomocą obiektu Deferred 409

- 13.1. Wprowadzenie do obietnic 410
- 13.2. Obiekty Deferred i Promise 414
- 13.3. Metody obiektu Deferred 415
 - 13.3.1. Rozstrzyganie lub odrzucanie obiektu Deferred 415
 - 13.3.2. Wykonywanie funkcji podczas rozstrzygania lub odrzucania 417
 - 13.3.3. Metoda when() 420
 - 13.3.4. Powiadamianie o postępie procesu związanego z obiektem Deferred 423
 - 13.3.5. Śledzenie postępu 424
 - 13.3.6. Użycie obiektu Promise 426
 - 13.3.7. Krótkie omówienie metody then() 429
 - 13.3.8. Zawsze wykonuj procedurę obsługi zdarzeń 433
 - 13.3.9. Określanie stanu obiektu Deferred 433
- 13.4. Stosowanie obietnic dla wszystkiego 434
- 13.5. Podsumowanie 436

Rozdział 14. Wykonywanie testów jednostkowych za pomocą biblioteki QUnit 437

- 14.1. Dlaczego testowanie odgrywa ważną rolę? 438
 - 14.1.1. Dlaczego testowanie jednostkowe? 439
 - 14.1.2. Środowiska testowania jednostkowego kodu JavaScript 441
- 14.2. Pierwsze kroki ze środowiskiem QUnit 442
- 14.3. Tworzenie testów na potrzeby kodu synchronicznego 445
- 14.4. Testowanie kodu za pomocą asercji 446
 - 14.4.1. Metody equal(), strictEqual(), notEqual() i notStrictEqual() 447
 - 14.4.2. Inne metody asercji 450
 - 14.4.3. Metoda asercji throws() 451
- 14.5. Testowanie zadań asynchronicznych 453
- 14.6. Opcje noglobals i notrycatch 455
- 14.7. Grupowanie testów w modułach 456
- 14.8. Konfigurowanie środowiska QUnit 458
- 14.9. Przykładowy pakiet testów 459
- 14.10. Podsumowanie 463

Rozdział 15. Wykorzystanie biblioteki jQuery w dużych projektach 465

- 15.1. Zwiększanie wydajności selektorów 467
 - 15.1.1. Unikanie selektora uniwersalnego 467
 - 15.1.2. Ulepszanie selektora klasy 468
 - 15.1.3. Nie nadużywaj parametru context 468
 - 15.1.4. Optymalizowanie filtrów 469
 - 15.1.5. Nie określaj przesadnie selektorów 471
- 15.2. Organizowanie kodu w modułach 471
 - 15.2.1. Wzorzec oparty na literalach obiektowych 472
 - 15.2.2. Wzorzec oparty na modułach 473
- 15.3. Ładowanie modułów za pomocą biblioteki RequireJS 475
 - 15.3.1. Rozpoczęcie pracy z biblioteką RequireJS 475
 - 15.3.2. Użycie biblioteki RequireJS z biblioteką jQuery 477
- 15.4. Zarządzanie zależnościami za pomocą narzędzia Bower 479
 - 15.4.1. Rozpoczęcie pracy z narzędziem Bower 479
 - 15.4.2. Wyszukiwanie pakietu 481
 - 15.4.3. Instalowanie, aktualizowanie i usuwanie pakietów 481
- 15.5. Tworzenie aplikacji z jedną stroną za pomocą środowiska Backbone.js 483
 - 15.5.1. Dlaczego należy używać środowiska MV*? 485
 - 15.5.2. Rozpoczęcie pracy ze środowiskiem Backbone.js 485
 - 15.5.3. Tworzenie aplikacji menedżera zadań do wykonania za pomocą środowiska Backbone.js 490
- 15.6. Podsumowanie 500
- 15.7. Zakończenie 502

DODATKI 503**Dodatek A Język JavaScript, jaki musisz znać, choć być może tak nie jest 505**

- A.1. Podstawowe informacje o typie danych Object języka JavaScript 506
 - A.1.1. W jaki sposób powstają obiekty? 506
 - A.1.2. Właściwości obiektów 507
 - A.1.3. Literały obiektowe 509
 - A.1.4. Obiekty jako właściwości obiektu window 510
- A.2. Funkcje jako „obywatele pierwszej klasy” 511
 - A.2.1. Wyrażenia funkcji i ich deklaracje 511
 - A.2.2. Funkcje jako wywołania zwrotne 513
 - A.2.3. Jakie jest przeznaczenie słowa kluczowego this? 514
 - A.2.4. Domknięcia 518
 - A.2.5. Wzorzec IIFE 520
- A.3. Podsumowanie 522

Skorowidz 523

1

Wprowadzenie do biblioteki jQuery

W rozdziale omówiono następujące zagadnienia:

- Dokładne przeznaczenie biblioteki jQuery i powody, dla których należy z niej korzystać.
- Strategia przejrzystości kodu JavaScript.
- Wybór właściwej wersji biblioteki jQuery.
- Podstawowe elementy biblioteki jQuery i związane z nią pojęcia.

„Istnieją tylko dwa rodzaje języków: takie, na które ludzie narzekają, oraz takie, z których nikt nie korzysta”. Zdanie to, wypowiedziane przez Bjarne’a Stroustrupa, który zaprojektował i zaimplementował język C++, bardzo dobrze podsumowuje odczucia związane z językiem JavaScript. Nad tym, a także nad kilkoma innymi językami (a szczególnie językiem PHP) „lamentowano” przez kilka lat, uważając je za „złe” języki. Później stało się coś magicznego. Dzięki pojawieniu się technologii Ajax, udostępnieniu kilku bibliotek, takich jak Prototype, Moo Tools i jQuery, a także nowych aplikacji internetowych o dużym poziomie interaktywności (być może spotkałeś się również z określaniem ich mianem *aplikacji z pojedynczą stroną*) projektanci zaczęli uzmysławiać sobie potencjał języka JavaScript. Obecnie język ten jest również jednym z najbardziej wszechobecnych dzięki środowisku Node.js, czyli platformie, która umożliwia użycie języka JavaScript jako języka serwerowego, a także dzięki środowisku PhoneGap służącemu do tworzenia hybrydowych aplikacji dla urządzeń przenośnych.

Biblioteka jQuery to darmowa (objęta licencją MIT License) i popularna biblioteka języka JavaScript stworzona przez Johna Resiga w 2006 r. Została zaprojektowana w celu uproszczenia tworzenia skryptów HTML po stronie klienta. Jak stwierdzono w witrynie internetowej biblioteki jQuery, jest to szybka, niewielka i bogata w możliwości biblioteka języka JavaScript. Dzięki łatwemu w użyciu interfejsowi API, który współpracuje z wieloma przeglądarkami, biblioteka znacznie upraszcza wykonywanie takich operacji jak przechodzenie w obrębie dokumentów HTML i modyfikowanie ich, tworzenie animacji oraz obsługiwanie zdarzeń i technologii Ajax. W połączeniu z wszechstronnością i rozszerzalnością biblioteka jQuery zmieniła sposób, w jaki miliony osób tworzą kod JavaScript.

Choć treść powyższego stwierdzenia możesz uznać za mającą charakter trochę autopromocyjny lub „arogancki”, zawiera ona samą prawdę. Biblioteka jQuery *naprawdę* zmieniła to, jak miliony projektantów pisze swój kod. Zastosowanie tej biblioteki jest tak powszechne, że zgodnie z danymi statystycznymi serwisu BuiltWith (z kwietnia 2015 r.) biblioteka jQuery używana jest w przypadku 63% miliona czołowych witryn internetowych (<http://trends.builtwith.com/javascript/jquery>). Wspomniana wcześniej biblioteka Moo Tools, która jest głównym konkurentem biblioteki jQuery, wykorzystywana jest jedynie w wypadku 3% witryn (<http://trends.builtwith.com/javascript/MooTools>), natomiast biblioteka Prototype może się „pochwalić” wynikiem wynoszącym zaledwie 2,5% (<http://trends.builtwith.com/javascript/Prototype>).

Biblioteka jQuery używana jest przez część spośród najważniejszych na świecie firm i witryn internetowych, takich jak Microsoft, Amazon, Dell, Etsy, Netflix, Best Buy, Instagram, Fox News, GoDaddy oraz wiele innych. Jeśli miałeś jakiegokolwiek wątpliwości związane z tą biblioteką, te dane powinny przekonać Cię, że jest to stabilna i niezawodna biblioteka, której możesz użyć w swoich projektach.

W książce omówiono wiele aspektów biblioteki, począwszy od podstawowych pojęć, takich jak selektory i metody umożliwiające przechodzenie w obrębie modelu DOM (*Document Object Model*), a skończywszy na bardziej zaawansowanych zagadnieniach, takich jak rozszerzanie funkcjonalności (tworzenie dodatków), zwiększanie wydajności kodu i testowanie. Przyjęto, że dysponujesz znajomością języka JavaScript w minimalnym stopniu. Jeśli musisz sobie odświeżyć posiadaną wiedzę, zajrzyj do dodatku. Jeżeli nie jesteś zaznajomiony z językiem, możesz uznać treść książki za trudną, dlatego zachęcamy do poznania języka, a następnie powrócenia do tej książki. Poczekamy.

Czy wróciłeś? Miło Cię widzieć ponownie! Zacznijmy od początku, czyli od omówienia tego, co biblioteka jQuery ma do zaoferowania, a także tego, jak może pomóc w procesie projektowania aplikacji internetowych.

1.1. Pisz mniej, a rób więcej

Motto biblioteki jQuery brzmi następująco: „Pisz mniej, a rób więcej”. Jeśli kiedykolwiek poświęciłeś czas na podejmowanie prób dodania do utworzonych stron elementów dynamicznych, to prawdopodobnie stwierdziłeś, że wykonywanie prostych

zadań za pomocą czystego kodu JavaScript może wygenerować dziesiątki wierszy kodu. Twórca biblioteki jQuery napisał ją specjalnie z myślą o sprawieniu, że typowe zadania staną się trywialne i łatwe do opanowania, a ponadto umożliwią rozwiązanie problemów spowodowanych przez niezgodność przeglądarek.

Każdy, kto na przykład miał do czynienia z przyciskami opcji w kodzie JavaScript, wie, że mało ciekawym doświadczeniem jest stwierdzanie, jaki element przycisku opcji grupy takich przycisków jest w danym momencie zaznaczony, a także uzyskiwanie wartości atrybutu `value` takiego elementu. Grupa przycisków opcji wymaga zlokalizowania, a każdy element przycisku opcji wynikowego zestawu musi kolejno zostać poddany inspekcji w celu określenia, dla którego elementu ustawiono atrybut `checked`. Atrybut `value` takiego elementu może następnie zostać uzyskany.

Aby zapewnić zgodność z przeglądarką Internet Explorer 6 i jej nowszymi wersjami (jeśli zignorujesz niektóre starsze przeglądarki, istnieje lepsze rozwiązanie), odpowiedni kod można zaimplementować w następujący sposób:

```
var checkedValue;
var elements = document.getElementsByTagName('input');
for (var i = 0; i < elements.length; i++) {
    if (elements[i].type === 'radio' &&
        elements[i].name === 'some-radio-group' &&
        elements[i].checked) {
        checkedValue = elements[i].value;
        break;
    }
}
```

Powyższy kod porównaj ze sposobem osiągnięcia tego samego za pomocą biblioteki jQuery:

```
var checkedValue =
    jQuery('input:radio[name="some-radio-group"]:checked').val();
```

Nie martw się, jeśli kod wygląda teraz trochę tajemniczo. Wkrótce zrozumiesz, jakie jest jego działanie, a ponadto będziesz tworzyć własne krótkie, ale bogate w możliwości instrukcje biblioteki jQuery, aby zapewnić dynamikę swoim stronom. W tym miejscu zmierzamy do pokazania, jak biblioteka umożliwia zamianę mnóstwa wierszy kodu na tylko jeden wiersz.

To, co sprawia, że powyższa instrukcja biblioteki jQuery jest taka krótka, to duże możliwości *selektora*, czyli wyrażenia używanego do identyfikowania elementów docelowych na stronie. Selektor daje możliwość łatwego zlokalizowania i pobierania wymaganych elementów. W tym przypadku jest to wybrany element w grupie przycisków opcji. Jeśli nie pobrałeś jeszcze przykładowego kodu, będzie teraz znakomity moment, aby tak postąpić. Kod możesz pobrać pod adresem: . Po rozpakowaniu archiwum z kodem w przeglądarce załaduj stronę HTML znajdującą się w pliku *rozdzial_1/radio.group.html*. W przypadku tej strony (rysunek 1.1) do określenia, jaki został wybrany przycisk opcji, używana jest wcześniej zaprezentowana instrukcja biblioteki jQuery.

W omawianym przykładzie pokazano, jak prosty i zwięzły może być kod utworzony za pomocą biblioteki jQuery. Nie jest to jedyny atut tej biblioteki. Jeśli byłoby inaczej, dawno temu by się jej pozbyto. Niemniej jednak jedną z największych zalet biblioteki

Jaka jest Twoja odpowiedź? ☒ Tak ☐ Nie ☐ Być może ☐ Nie wiem

Wybrano element przycisku opcji o wartości **Tak**.

Rysunek 1.1. Określanie, jaki przycisk opcji został wybrany, jest proste do zrealizowania za pomocą jednej instrukcji z kodem biblioteki jQuery!

jQuery jest zapewnianiem przez nią możliwości uzyskiwania elementów za pomocą złożonych selektorów bez przejmowania się kwestią zgodności z różnymi przeglądarkami, a zwłaszcza ze starszymi.

Przy wykonywaniu operacji wybierania bazujesz na dwóch elementach, a mianowicie na metodzie i selektorze. Obecnie najnowsze wersje wszystkich podstawowych przeglądarek obsługują metody macierzyste służące do wybierania elementów, takie jak `document.querySelector()` i `document.querySelectorAll()`. Umożliwiają one zastosowanie bardziej złożonych selektorów zamiast zwykłej operacji wybierania w oparciu o identyfikator lub klasę.

Ponadto nowe selektory arkuszy stylów CSS3 są powszechnie obsługiwane przez nowoczesne przeglądarki. Jeśli zdecydowałbyś się obsługiwać tylko takie przeglądarki, a możliwości biblioteki jQuery związane byłyby wyłącznie z wybieraniem elementów, dostępna byłaby moc obliczeniowa wystarczająca do tego, aby pominąć obciążenie wynikające z zastosowania biblioteki w witrynie internetowej. To, że wiele osób nadal bazuje na starszych przeglądarkach, do których obsługi możesz być zmuszony, może być rzeczywistym utrudnieniem, gdyż konieczne jest zajmowanie się wszystkimi niespójnościami. Jest to jeden z głównych powodów wykorzystania biblioteki jQuery. Umożliwia ona użycie jej selektorów w niezawodny sposób bez obaw, że kod nie zadziała w przeglądarkach, które nie obsługują ich we własnym zakresie.

UWAGA Jeśli zastanawiasz się, jakie przeglądarki są obecnie uważane za nowoczesne, to są nimi: Internet Explorer w wersji 10 i nowszych, a także najnowsze wersje przeglądarek Chrome, Opera, Firefox i Safari.

Czy nadal nie jesteś przekonany? Pod adresem <http://goo.gl/eULyPT> dostępna jest lista problemów, którymi musisz się zająć we własnym zakresie, gdy nie używasz biblioteki jQuery. Jak już wspomniano, biblioteka zapewnia znacznie więcej możliwości, o których dowiesz się w pozostałej części książki.

Dowiedzmy się teraz, jak kod JavaScript powinien być używany na tworzonych stronach.

1.2. Przejrzysty kod JavaScript

Być może pamiętasz nie najlepsze czasy przed pojawieniem się arkuszy stylów CSS, gdy w obrębie stron HTML konieczne było łączenie znaczników określających styl ze znacznikami struktury dokumentu. Każdy, kto tworzył strony przez dowolny czas, z pewnością musiał tak postępować (czemu najprawdopodobniej nie towarzyszyło uczucie sympatii).

Dodanie arkuszy stylów CSS do zestawów narzędziowych używanych do projektowania aplikacji internetowych pozwala na oddzielenie informacji o stylach od struktury dokumentu, a farsom w rodzaju znacznika `<font>` zapewnia „naprawdę zasłużonego kopniaka”. Separacja stylu od struktury nie tylko ułatwia zarządzanie dokumentami, ale też oferuje wszechstronność pozwalającą na całkowitą zmianę renderowania strony pod względem stylistycznym, polegającą na wymianie różnych arkuszy stylów.

Niewielu z czytelników dobrowolnie cofnęłoby się do czasów, gdy style stosowano przy użyciu elementów HTML. Okazuje się, że ponadto w dalszym ciągu zbyt powszechne są znaczniki podobne do następujących:

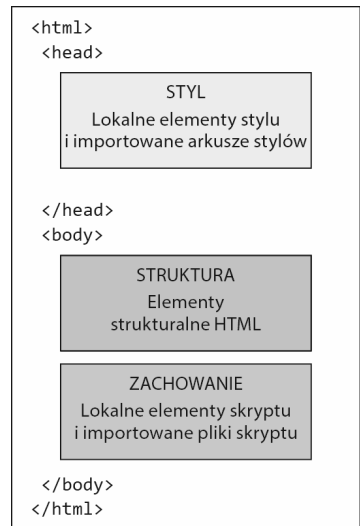
```
<button onclick="document.getElementById('xyz').style.color='red';">  
  Kliknij mnie  
</button>
```

Z łatwością możesz zauważyć, że styl tego elementu `button` nie jest stosowany przy użyciu znacznika `<font>` oraz innych przestarzałych znaczników powiązanych ze stylem. Styl elementu jest określany przez dowolne — jeśli zostały zdefiniowane — reguły arkuszy stylów CSS (ich kodu tutaj nie pokazano), które obowiązują na stronie. Choć taka deklaracja nie łączy znaczników *stylu* ze strukturą, wiąże z nią *zachowanie*. Deklaracja uwzględnia kod JavaScript wykonywany w momencie kliknięcia przycisku jako część znacznika elementu `button` za pośrednictwem atrybutu `onclick` (w tym przypadku atrybut zmienia na czerwony kolor elementu modelu DOM z identyfikatorem o wartości `xyz`). Sprawdźmy, jak w tej sytuacji można wprowadzić ulepszenia.

1.2.1. Oddzielanie zachowania od struktury

Z wszystkich tych samych powodów, dla których pożądane jest rozdzielenie stylu i struktury w obrębie dokumentu HTML, tak samo korzystne (jeśli nawet nie bardziej) jest odseparowanie *zachowania* od struktury. W idealnej sytuacji struktura strony HTML powinna być podobna do pokazanej na rysunku 1.2. W jej wypadku struktura, styl i zachowanie są ładnie wydzielone w obrębie własnego obszaru.

Strategia ta, określana mianem *przejrzystego kodu JavaScript*, jest obecnie uwzględniana przez każdą bardziej znaną bibliotekę języka JavaScript. Ułatwia to twórcom stron uzyskiwanie takiej przydatnej separacji w obrębie swoich stron. Rdzeń biblioteki jQuery, która spopularyzowała tę strategię, jest odpowiednio zoptymalizowany pod kątem łatwego generowania przejrzystego kodu JavaScript. W przypadku takiego kodu za niepoprawne uznane zostaną *dowolne* wyrażenia lub instrukcje języka JavaScript umieszczone w obrębie znaczników HTML w sekcji `<body>` stron



Rysunek 1.2. Dzięki przejrzystemu rozmieszczeniu w obrębie strony struktury, stylu i zachowania w osobnych obszarach maksymalnie zwiększana jest czytelność i możliwość utrzymania kodu

HTML, które to wyrażenia lub instrukcje są atrybutami elementów HTML (np. `onclick`) albo zawarte są w blokach skryptu zlokalizowanych w dowolnym miejscu innym niż sam koniec sekcji `<body>` strony.

Możesz zadać następujące pytanie: „Jak jednak mogę skonfigurować przycisk bez atrybutu `onclick`?”. Przyjrzyj się następującej modyfikacji elementu przycisku:

```
<button id="test-button">Kliknij mnie</button>
```

Dużo prościej! Teraz jednak zauważysz, że przycisk nie wykonuje żadnego działania. Możesz klikać przycisk przez cały dzień, a nie spowoduje to żadnego działania. Poprawmy to.

1.2.2. Oddzielanie skryptu

Zamiast osadzać w znacznikach przycisku informacje o jego zachowaniu, dokonasz oddzielenia skryptu przez przeniesienie go do bloku `script`. Postępując zgodnie z obowiązującymi obecnie najlepszymi praktykami, skrypt należy umieścić na dole strony przed zamykającym znacznikiem `</body>`:

```
<script>
  document.getElementById('test-button').addEventListener(
    'click',
    function() {
      document.getElementById('xyz').style.color = 'red';
    },
    false
  );
</script>
```

Ponieważ skrypt umieszczasz u dołu strony, nie musisz używać procedury obsługi powiązanej ze zdarzeniem `onload` obiektu `window`, po jaką (niesłusznie) w przeszłości sięgali projektanci, albo czekać na zdarzenie `DOMContentLoaded`, które dostępne jest tylko w nowoczesnych przeglądarkach. Zdarzenie to jest aktywowane w momencie całkowitego załadowania i przeanalizowania dokumentu HTML bez oczekiwania na zakończenie ładowania arkuszy stylów, obrazów itp. Zdarzenie `load` wyzwalane jest w chwili ukończenia ładowania strony HTML oraz jej zasobów zależnych (do tego tematu powrócimy w punkcie 1.5.3). Dzięki umieszczeniu skryptu na dole strony, gdy przeglądarka analizuje instrukcję, element `button` istnieje, ponieważ jego znaczniki zostały poddane analizie. Oznacza to, że możesz bezpiecznie element rozszerzyć.

UWAGA Ze względów wydajnościowych elementy `script` zawsze powinny być umieszczane na dole treści dokumentu. Pierwszym powodem jest umożliwienie renderowania progresywnego, a drugim zapewnienie zwiększenia równoległości operacji pobierania. Uzasadnieniem pierwszego powodu jest to, że renderowanie jest blokowane dla całej treści znajdującej się poniżej elementu `script`. Drugi powód wynika z tego, że przeglądarka nie rozpocznie żadnych innych operacji pobierania (nawet w wypadku innej nazwy hosta), jeśli pobierane są dane elementu `script`.

Zamieszczony na poprzedniej stronie fragment kodu jest kolejnym przykładem, który nie jest całkowicie zgodny z przeglądarkami, pod kątem jakich może być realizowany projekt. W kodzie tym użyto metody `addEventListener()` języka JavaScript, która nie jest obsługiwana przez przeglądarkę Internet Explorer w wersjach od 6 do 8. Jak się okaże w dalszej części książki, biblioteka jQuery okazuje się pomocna przy rozwiązywaniu również tego problemu.

Choć strategia przejrzystego kodu JavaScript to bogata w możliwości technika pozwalająca na wyraźne oddzielenie zakresów odpowiedzialności w obrębie aplikacji internetowej, nie jest pozbawiona niedogodności. Być może zauważyłeś już, że osiągnięcie celu wymagało kilku wierszy skryptu więcej niż w sytuacji, gdy kod umieszczono w znacznikach przycisku. Przejrzysty kod JavaScript może spowodować zwiększenie liczby wierszy kodu skryptu niezbędnego do utworzenia, a ponadto wymaga pewnej dyscypliny oraz stosowania właściwych wzorców tworzenia kodu dla skryptu po stronie klienta.

Nie ma w tym jednak niczego złego. Wskazane jest wszystko, co przekonuje do pisania kodu po stronie klienta na takim samym poziomie dbałości, jaki zwykle towarzyszy tworzeniu kodu po stronie serwera! *Oznacza to* jednak dodatkowy nakład pracy — oczywiście gdy nie skorzysta się z biblioteki jQuery.

W wypadku biblioteki jQuery szczególnie skupiono się na sprawieniu, że tworzenie kodu stron przy użyciu technik przejrzystego kodu JavaScript będzie łatwym i przyjemnym zadaniem, które nie wymaga ponoszenia sporej ceny w postaci nakładu pracy lub ilości kodu. Stwierdzisz, że efektywne korzystanie z tej biblioteki umożliwi osiągnięcie znacznie więcej na stronach, gdy tworzona jest *mniej* ilość kodu. Czyż nie obowiązuje nadal motto „Pisz mniej, a rób więcej”? Bez zbędnego dalszego rozpisywania się zaczniemy wyjaśnianie, w jaki sposób biblioteka jQuery tak bardzo ułatwia dodawanie zaawansowanych funkcji do stron bez oczekiwanych utrudnień.

1.3. Instalowanie biblioteki jQuery

Gdy już wiesz, czym jest biblioteka jQuery i jakie są jej możliwości, to aby rozpocząć korzystanie z niej, musisz ją pobrać. W tym celu odwiedź stronę o adresie <http://jquery.com/download/>. Po wyświetleniu jej prawdopodobnie zostaniesz przytłoczony mnóstwem dostępnych opcji. Gałąź wersji 1.x, 2.x lub 3.x? Skompresowane czy bez kompresji? Czy pobierać bibliotekę, czy skorzystać z sieci CDN (*Content Delivery Network*)? Wybór zależy od kilku czynników. Abyś mógł dokonać świadomego wyboru, zaprezentujemy różnice.

1.3.1. Wybieranie właściwej wersji

W kwietniu 2013 r. zespół tworzący bibliotekę jQuery zaprezentował jej wersję 2.0 z zamiarem wzięcia pod uwagę przyszłości technologii internetowych zamiast ich przeszłości, zwłaszcza z punktu widzenia przeglądarek. Do tego momentu biblioteka jQuery obsługiwała wszystkie najnowsze wersje przeglądarek Chrome, Firefox, Safari, Opera oraz Internet Explorer (począwszy od wersji 6). Wraz z pojawieniem się wersji 2.0 zespół zdecydował się na zrezygnowanie z obsługi starszych wersji 6, 7 i 8 przeglądarki

Internet Explorer, aby skoncentrować się na przyszłości technologii internetowych, a nie na ich przeszłości.

Decyzja ta spowodowała usunięcie porcji kodu utworzonego w celu eliminowania niezgodności przeglądarek oraz radzenia sobie z funkcjami nieobecnymi w tych „prehistorycznych” przeglądarkach. Zrealizowanie tego zadania przyczyniło się do uzyskania mniejszej (o 12%) i szybszej bazy kodu. Choć wersje 1.x i 2.x to dwie różne gałęzie, są ze sobą ściśle powiązane. Występuje równość funkcji w wypadku wersji biblioteki jQuery 1.10 i 2.0, wersji 1.11 i 2.1 itd.

W październiku 2014 r. Dave Methvin, przewodniczący fundacji jQuery Foundation (zajmuje się biblioteką jQuery oraz innymi projektami; <https://jquery.org/>) opublikował post na blogu (<http://blog.jquery.com/2014/10/29/jquery-3-0-the-next-generations/>), w którym publicznie ogłosił plan udostępnienia nowej głównej wersji biblioteki jQuery, czyli jQuery 3. Tak samo jak wersja 1.x obsługuje stare przeglądarki, a wersja 2.x dotyczy nowoczesnych przeglądarek, tak biblioteka jQuery 3 zapewnia dwa warianty. Warianty jQuery Compat 3 i jQuery 3 to odpowiednio następcy wersji 1.x i 2.x. Oto dodatkowe wyjaśnienia przewodniczącego:

Począwszy od tych wersji, będziemy też zmieniać naszą strategię pod kątem obsługi przeglądarek. Główny pakiet biblioteki jQuery pozostaje niewielki i zwarty dzięki obsługiwaniu sprawdzonych przeglądarek (bieżące i poprzednie wersje konkretnej przeglądarki), które są powszechnie używane w momencie udostępnienia pakietu. Możliwa jest obsługa dodatkowych przeglądarek w tym pakiecie na podstawie ich udziału w rynku. Pakiet wariantu jQuery Compat oferuje znacznie większe wsparcie przeglądarek, co jednak powoduje większy rozmiar pliku i potencjalnie niższą wydajność.

Wraz z udostępnieniem nowej wersji zespół skorzystał z możliwości zrezygnowania z obsługi niektórych przeglądarek, usunięcia wielu błędów i ulepszenia kilku funkcji.

Pierwsza kwestia do rozważenia przy podejmowaniu decyzji, jaka wersja ma zostać użyta, dotyczy tego, jakie przeglądarki muszą być obsługiwane w ramach projektu. W tabeli 1.1 opisano przeglądarki obsługiwane przez każdą główną wersję biblioteki jQuery.

Jak widać w tabeli, występuje pewien stopień pokrywania się pod względem obsługiwanych wersji przeglądarek. Miej jednak na uwadze, że to, co reprezentowane jest przez określenie „bieżące i poprzednie wersje” (wskazuje na bieżące i poprzednie wersje przeglądarki stosowane w momencie opublikowania nowej wersji biblioteki jQuery), zmienia się zależnie od daty udostępnienia nowej wersji biblioteki jQuery.

Inną istotną kwestią, jaką trzeba uwzględnić przy podejmowaniu decyzji, jest to, gdzie skorzystasz z biblioteki jQuery. Oto niektóre przypadki użycia, które mogą ułatwić dokonanie wyboru:

- Witryny internetowe, które nie wymagają obsługi starszych wersji przeglądarek Internet Explorer i Opera, a także innych przeglądarek korzystających z gałęzi 3.x. Dotyczy to witryn internetowych uruchomionych w środowisku kontrolowanym, takim jak firmowa sieć lokalna.

Tabela 1.1. Zestawienie przeglądarek obsługiwanych przez główne wersje biblioteki jQuery

Przeglądarki	jQuery 1	jQuery 2	jQuery Compat 3	jQuery 3
Internet Explorer	6+	9+	8+	9+
Chrome	Bieżące i poprzednie wersje	Bieżące i poprzednie wersje	Bieżące i poprzednie wersje	Bieżące i poprzednie wersje
Firefox	Bieżące i poprzednie wersje	Bieżące i poprzednie wersje	Bieżące i poprzednie wersje	Bieżące i poprzednie wersje
Safari	5.1+	5.1+	7.0+	7.0+
Opera	12.1x Bieżące i poprzednie wersje	12.1x Bieżące i poprzednie wersje	Bieżące i poprzednie wersje	Bieżące i poprzednie wersje
iOS	6.1+	6.1+	7.0+	7.0+
Android	2.3 4.0+	2.3 4.0+	2.3 4.0+	2.3 4.0+

- Witryny internetowe, których treść musi być kierowana do jak największej grupy odbiorców (np. witryna rządowa), powinny korzystać z gałęzi 1.x.
- Jeśli projektujesz witrynę internetową, która musi być dostosowana do wymagań szerszej grupy odbiorców, lecz bez konieczności obsługi wersji 6 i 7 przeglądarek Internet Explorer oraz starszych wersji przeglądarek Opera i Safari, należy zastosować bibliotekę jQuery Compat 3.x.
- Jeśli nie wymagasz obsługi przeglądarki Internet Explorer w wersji 8 lub starszej, ale konieczne jest wsparcie starych przeglądarek Opera i Safari, należy użyć biblioteki jQuery 2.x.
- Aplikacje dla urządzeń przenośnych zaprojektowane za pomocą środowiska PhoneGap lub podobnych środowisk mogą korzystać z biblioteki jQuery 3.x.
- Aplikacje dla systemu Firefox OS lub Chrome OS mogą używać biblioteki jQuery 3.x.
- W wypadku witryn internetowych, które oparte są na bardzo starych dodatkach, może być wymagane zastosowanie biblioteki jQuery 1.x (zależnie od faktycznego kodu dodatków).

Podsumowując, dwie kwestie do rozważenia to miejsce zastosowania biblioteki oraz to, jakie przeglądarki mają być obsługiwane.

Innym powodem nieporozumień może być wybór między wersją skompresowaną (określaną też mianem *zminimalizowanej*) przewidzianą do użycia w fazie produkcyjnej a wersją bez kompresji przeznaczoną do zastosowania w fazie projektowania (na rysunku 1.3 zaprezentowano porównanie obu wersji). Zaletą biblioteki w wersji zminimalizowanej jest zmniejszenie wielkości pliku, dzięki czemu użytkownikom zapewnia się mniejsze wykorzystanie dostępnej przepustowości połączenia sieciowego. Redukcja wielkości pliku jest osiągnięta przez usuwanie zbędnych obszarów (*wcięcia w kodzie*) i komentarzy do kodu, które są przydatne dla projektantów, ale ignorowane

przez silniki języka JavaScript, a także przez skracanie nazw zmiennych (*zaciemnianie kodu*). Zmiany te powodują uzyskanie kodu, który jest mniej czytelny i trudniejszy do poddania debugowaniu (z tego właśnie powodu nie należy używać tej wersji kodu w fazie projektowania), ale o mniejszej wielkości.

Bez kompresji	<pre>// Obsługa w momencie zapewnienia gotowości modelu DOM ready: function(wait) { // Przerwanie, jeśli istnieją oczekujące blokady lub zapewniono już gotowość if (wait === true ? --jQuery.readyWait : jQuery.isReady) { return; } // Sprawdzenie istnienia treści, przynajmniej w wypadku przeglądarki IE, // okazuje się trochę zbędne (etykieta #5443). if (!document.body) { return setTimeout(jQuery.ready); } // Zapamiętanie gotowości modelu DOM jQuery.isReady = true; // Jeśli aktywowano zdarzenie gotowości modelu DOM, ma miejsce dekrementacja, // a w razie potrzeby oczekiwanie if (wait !== true && --jQuery.readyWait > 0) { return; } // W przypadku powiązania funkcji do wykonania readyList.resolveWith(document, [jQuery]); // Aktywowanie wszystkich powiązanych zdarzeń gotowości if (jQuery.fn.triggerHandler) { jQuery(document).triggerHandler("ready"); jQuery(document).off("ready"); } } }</pre>
Z kompresją	<pre>ready: function(a){if(a===!0?!--m.readyWait:!m.isReady){if(!y.body)return setTimeout(m.ready);m.isReady=!0,a!==!0&&--m.readyWait>0 H.resolveWith(y, [m],m.fn.triggerHandler&&(m(y).triggerHandler("ready"),m(y).off("ready")))}}}</pre>

Rysunek 1.3. U góry pokazano fragment kodu pochodzący z kodu źródłowego biblioteki jQuery prezentujący wersję bez kompresji. Na dole widoczny jest ten sam fragment kodu w wersji zminimalizowanej na potrzeby użycia do celów produkcyjnych

W książce będziemy bazować na bibliotece jQuery 1.x, aby umożliwić testowanie tworzonego kodu w największej możliwej grupie przeglądarek. Zwrócimy jednak uwagę na wszystkie różnice, jakie pojawiły się w bibliotece jQuery 3, dzięki czemu Twoja wiedza będzie aktualna w maksymalnym możliwym stopniu.

Wybór właściwej wersji biblioteki jQuery jest istotny, ale wspomniano też o różnicy między lokalnym udostępnianiem tej biblioteki a wykorzystaniem do tego sieci CDN.

1.3.2. Zwiększanie wydajności przy użyciu sieci CDN

Obecnie powszechną praktyką jest udostępnianie plików takich jak obrazy i biblioteki za pośrednictwem sieci dostarczania treści (CDN — *Content Delivery Network*), co ma na celu zwiększenie wydajności witryny internetowej. Sieć CDN to system rozproszony serwerów utworzony do oferowania treści przy zapewnieniu wysokiej dostępności i wydajności. Możesz być świadomy tego, że przeglądarki mają możliwość jed-

noczesnego pobierania z hosta ustalonego zestawu treści pochodzącej zwykle z liczby plików wynoszącej od czterech do ośmiu. Ponieważ pliki udostępniane za pomocą sieci CDN są zapewniane z innego hosta, możesz przyspieszyć cały proces ładowania przez zwiększenie liczby jednocześnie pobieranych plików. Poza tym wiele witryn internetowych korzysta obecnie z sieci CDN, dlatego jest duże prawdopodobieństwo, że wymagana biblioteka będzie się już znajdować w pamięci podręcznej przeglądarki użytkownika. Zastosowanie sieci CDN do załadowania biblioteki jQuery nie gwarantuje lepszej wydajności w każdej sytuacji, gdyż wpływ ma na to wiele czynników. Radzimy sprawdzić, jaka konfiguracja będzie najlepiej się sprawdzać w konkretnej sytuacji.

Choć dostępnych jest obecnie kilka sieci CDN, które mogą zostać użyte do dołączania biblioteki jQuery, do najbardziej niezawodnych zaliczają się następujące: jQuery CDN (<http://code.jquery.com/>), Google CDN (<https://developers.google.com/speed/libraries/devguide>) oraz Microsoft CDN (<http://www.asp.net/ajaxlibrary/cdn.ashx>).

Żałómy, że chcesz dołączyć skompresowaną wersję biblioteki jQuery 1.11.3 za pomocą sieci jQuery CDN. W tym celu możesz utworzyć następujący wiersz kodu:

```
<script src="//code.jquery.com/jquery-1.11.3.min.js"></script>
```

Jak być może już zauważyłeś, kod ten nie określa protokołu, jaki ma zostać użyty (HTTP lub HTTPS). Zamiast tego stosowany jest ten sam protokół, którego użyto w witrynie internetowej. Miej jednak świadomość, że wykorzystanie tej techniki w wypadku strony, która nie jest uruchomiona na serwerze WWW, spowoduje błąd.

Zastosowanie sieci CDN nie jest pozbawione mankamentów. Żaden serwer ani sieć nie zapewni nieustannej dostępności w internecie. Sieci CDN nie są tutaj wyjątkiem. Jeśli przy ładowaniu biblioteki jQuery bazujesz na sieci CDN, to w rzadkich sytuacjach, gdy jest ona niedostępna lub uległa awarii, a ponadto przeglądarka osoby odwiedzającej witrynę nie zawiera buforowanej kopii, kod witryny przestanie poprawnie działać. Może to być problemem w przypadku kluczowych aplikacji. Aby tego uniknąć, można zastosować dostępne proste i sprytne rozwiązanie, z którego korzysta wielu projektantów. I tym razem chcesz dołączyć zminimalizowaną wersję biblioteki jQuery 1.11.3, lecz teraz użyjesz następującego sprytnego rozwiązania:

```
<script src="//code.jquery.com/jquery-1.11.3.min.js"></script>
<script>window.jQuery || document.write('<script src="javascript/jquery-1.11.3.min.js">
↵</script>');</script>
```

Ideą działania tego kodu jest zażądanie kopii biblioteki z sieci CDN i określenie, czy biblioteka została załadowana przez sprawdzenie, czy zdefiniowano właściwość jQuery obiektu window. Jeśli test zakończy się niepowodzeniem, zostanie „wstrzyknięty” kod, który załaduje lokalnie dostępną kopię biblioteki. W tym konkretnym przypadku kopia ta jest przechowywana w folderze o nazwie javascript. Jeśli istnieje właściwość jQuery, spokojnie możesz skorzystać z metod biblioteki jQuery bez potrzeby ładowania lokalnie dostępnej kopii.

Test obecności właściwości jQuery wynika z tego, że jest ona dodawana po załadowaniu biblioteki. W tej właściwości możesz znaleźć wszystkie metody i inne właściwości biblioteki. Sugerujemy, aby w czasie projektowania korzystać z lokalnej kopii biblioteki jQuery, co pozwoli zapobiec wszelkim problemom z połączeniem.

Oprócz właściwości jQuery znajdziesz również skrót o nazwie \$, z którym bardzo często spotkasz się w tej książce i nie tylko w niej. Choć może się to wydać dziwne, w języku JavaScript dozwolona jest zmienna lub właściwość o nazwie \$. Znak \$ nazwano skrótem, ponieważ właściwie jest to ten sam obiekt właściwości jQuery, co potwierdza następująca instrukcja pochodząca z kodu źródłowego:

```
window.jQuery = window.$ = jQuery;
```

Do tej pory dowiedziałeś się, jak dołączyć bibliotekę jQuery do stron internetowych. Nie wiesz jednak nic na temat jej struktury. Zajmiemy się tym w następnym podrozdziale.

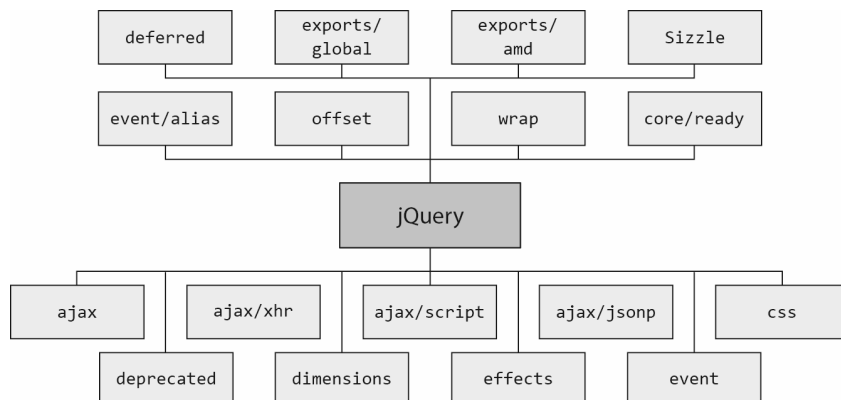
1.4. Struktura biblioteki jQuery

Repozytorium biblioteki jQuery (<https://github.com/jquery/jquery>) dostępne w witrynie GitHub to znakomity przykład tego, jak przez ostatnie lata zmieniał się projektowanie interfejsu. Choć nie jest to ściśle powiązane z użyciem samej biblioteki, zawsze istotna jest świadomość tego, jak doświadczeni projektanci organizują sobie przepływ pracy oraz wykorzystywane przez siebie narzędzia.

Jeśli jesteś doświadczonym projektantem interfejsów, może się okazać, że znasz już część tych narzędzi (jeśli nie wszystkie). Niemniej jednak zawsze warto odświeżyć sobie informacje na ten temat. Zespół projektantów wykorzystał na potrzeby prac związanych z rozwojem biblioteki jQuery najnowsze i najciekawsze technologie, jakie tworzą obecnie panoramę interfejsów użytkownika. Dokładniej rzecz biorąc, są to następujące technologie:

- *Node.js* (<http://nodejs.org/>) — platforma oparta na środowisku wykonawczym JavaScript przeglądarki Chrome, która umożliwia uruchamianie kodu JavaScript po stronie serwera.
- *npm* (<https://npmjs.org/>) — oficjalny menedżer pakietów platformy Node.js używany do instalowania pakietów, takich jak Grunt wraz z jego zadaniami.
- *Grunt* (<http://gruntjs.com/>) — narzędzie do wykonywania zadań, które automatyzuje typowe i powtarzające się zadania, takie jak budowanie, testowanie i minimalizowanie.
- *Git* (<http://git-scm.com/>) — darmowy, rozproszony system kontroli wersji, który śledzi zmiany wprowadzane w kodzie. Ułatwia on współpracę projektantów.

Kod źródłowy biblioteki jQuery jest też zgodny z formatem definicji modułu asynchronicznego (AMD — *Asynchronous Module Definition*). Format ten to propozycja dotycząca definiowania modułów, w przypadku których asynchronicznie może być ładowany zarówno moduł, jak i jego zależności. W praktyce oznacza to, że choć biblioteka jest używana jako unikatowy, pojedynczy blok, jej kod źródłowy dzielony jest na kilka plików (modułów), co pokazano na rysunku 1.4. Zależnościami powiązanymi z tymi plikami zarządza się z wykorzystaniem menedżera zależności (w tym wypadku jest to narzędzie RequireJS).



Rysunek 1.4. Schemat reprezentujący moduły biblioteki jQuery: ajax, ajax/xhr, ajax/script, ajax/jsonp, css, deprecated, dimensions, effects, event, event/alias, offset, wrap, core/ready, deferred, exports/global, exports/amd i Sizzle

Abyś mógł zorientować się, co jest wewnątrz modułów, prezentujemy kilka następujących przykładów:

- ajax — zawiera wszystkie funkcje technologii Ajax, takie jak `ajax()`, `get()` i `post()`.
- deprecated — zawiera wszystkie obecnie przestarzałe metody, które nie zostały usunięte. To, co znajduje się w tym module, zależy od wersji biblioteki jQuery.
- effects — zawiera metody, takie jak `animate()` i `slideUp()`, które umożliwiają tworzenie animacji.
- event — zawiera metody, takie jak `on()` i `off()`, które służą do dołączania do zdarzeń przeglądarki procedur obsługi zdarzeń.

Organizacja kodu źródłowego oparta na modułach zapewnia kolejną korzyść, czyli możliwość zbudowania wersji niestandardowej biblioteki jQuery, która zawiera wyłącznie wymagane moduły.

1.4.1. Tworzenie własnej wersji niestandardowej zmniejszającej wielkość pliku

Biblioteka jQuery oferuje możliwość budowania jej wersji niestandardowej, która zawiera tylko niezbędne funkcjonalności. Pozwala to zredukować wielkość pliku biblioteki. To z kolei prowadzi do zwiększenia wydajności, gdyż użytkownik musi pobrać mniej kilobajtów danych.

Istotna jest możliwość wyeliminowania zbędnych modułów. Choć możesz pomyśleć, że będziesz wymagać wszystkich możliwości oferowanych przez bibliotekę jQuery, wątpliwe jest, że w ramach tej samej witryny internetowej skorzystasz z wszystkich jej funkcji. Dlaczego więc nie usunąć tych zbędnych wierszy kodu, aby poprawić wydajność witryny internetowej?

Narzędzie Grunt umożliwia utworzenie wersji niestandardowej. Wyobraź sobie, że wymagasz zminimalizowanej wersji biblioteki jQuery 1.11.3 z wszystkimi funkcjonalnościami (z wyjątkiem przestarzałych metod i właściwości) i efektami. Aby zrealizować to zadanie, na komputerze musisz zainstalować narzędzia Node.js, Git i Grunt.

Po ich zainstalowaniu musisz powielić repozytorium biblioteki jQuery, uruchamiając następujące polecenie przy użyciu interfejsu wiersza poleceń CLI (*Command Line Interface*):

```
git clone git://github.com/jquery/jquery.git
```

Po ukończeniu procesu powielania wprowadź jeszcze następujące dwa polecenia:

```
npm install  
grunt custom:-deprecated,-effects
```

I to tyle! W folderze o nazwie *dist* znajdziesz wersję niestandardową biblioteki jQuery zarówno w wariancie zminimalizowanym, jak i w wariancie bez minimalizacji.

Takie rozwiązanie nie jest jednak pozbawione mankamentów. Pierwszy problem pojawia się w momencie opublikowania nowej wersji biblioteki jQuery. Drugi problem występuje, gdy nowa funkcjonalność witryny internetowej wymaga uwzględnienia w module składnika, który wcześniej nie został dołączony. W tych sytuacjach konieczne jest ponowne wykonanie wyżej podanych kroków (zwykle tylko poleceń) w celu utworzenia nowej wersji niestandardowej zawierającej nowe metody, poprawki błędów lub brakujący moduł.

Gdy już wiesz, jak umieścić bibliotekę w odpowiednim miejscu, a także jak utworzyć wersję niestandardową, pora poznać fundamenty biblioteki jQuery.

1.5. Fundamenty biblioteki jQuery

U swoich podstaw biblioteka jQuery koncentruje się na uzyskiwaniu elementów ze stron HTML i wykonywaniu na nich operacji. Jeśli jesteś zaznajomiony z arkuszami stylów CSS, masz już świadomość możliwości selektorów, które opisują grupy elementów według ich typu, atrybutów, umiejscowienia w dokumencie oraz innych kryteriów. W wypadku biblioteki jQuery możesz wykorzystać tę wiedzę oraz poziom możliwości do znacznego uproszczenia kodu JavaScript.

W przypadku biblioteki jQuery duży priorytet ma zapewnienie, że kod będzie działał spójnie we wszystkich najważniejszych przeglądarkach. Wiele spośród poważniejszych problemów związanych z kodem JavaScript zostało automatycznie rozwiązanych. Jeśli uznasz, że biblioteka jQuery wymaga jakiejś dodatkowej modyfikacji, to pamiętaj, że oferuje ona prosty, lecz bogaty w możliwości sposób rozszerzania jej funkcjonalności za pomocą dodatków, które szczegółowo zostaną omówione w rozdziale 12.

Na początek zajmijmy się samym obiektem jQuery, a także tym, w jaki sposób wiedzę na temat arkusza stylów CSS możesz wykorzystać do tworzenia zaawansowanego, a przy tym zwięzłego kodu.

1.5.1. Właściwości, narzędzia i metody

Jak wcześniej wspomniano, biblioteka jQuery jest dostępna za pośrednictwem właściwości o nazwie jQuery i skrótu \$. Użycie ich zapewnia dostęp do właściwości, metod i funkcji oferowanych przez bibliotekę jQuery.

`fx.off` to jedna z właściwości ujawnianych przez właściwość jQuery. Właściwość `fx.off` umożliwia włączanie lub wyłączanie efektów stosowanych za pomocą metod biblioteki jQuery. Ta oraz inne właściwości zostaną szczegółowo omówione w rozdziale 9.

Znacznie bardziej ekscytujące są *narzędzia*, które są też określane mianem *funkcji narzędziowych*. Możesz je traktować jako zestaw powszechnie używanych funkcji ogólnego przeznaczenia, które dołączono do biblioteki. Możesz stwierdzić, że biblioteka pełni względem nich rolę *przestrzeni nazw*.

Aby ogólnie zorientować się, czym są narzędzia, przyjrzyjmy się przykładowi. Jednym z dostępnych narzędzi jest funkcja służąca do przycinania łańcuchów. Jej zadaniem jest usuwanie białych znaków z początku i końca łańcucha. Wywołanie tej funkcji może wyglądać następująco:

```
var trimmed = $.trim(someString);
```

Jeśli wartością argumentu `someString` jest łańcuch " witaj ", wynikiem wywołania `$.trim()` będzie łańcuch "witaj". Jak widać, w tym przykładzie użyto skrótu biblioteki jQuery w postaci znaku `$`. Pamiętaj, że jest to taki identyfikator, jak dowolny inny w języku JavaScript. Utworzenie wywołania tej samej funkcji za pomocą identyfikatora jQuery, a nie jego aliasu, spowoduje uzyskanie następującego kodu:

```
var trimmed = jQuery.trim(someString);
```

Kolejnym przykładem funkcji narzędziowej jest funkcja `$.isArray()`, która — jak możesz się domyślić — sprawdza, czy dany argument jest tablicą.

Oprócz właściwości i funkcji biblioteka zapewnia też metody, które są dostępne po wywołaniu funkcji `jQuery()`. Dowiedzmy się więcej na jej temat.

1.5.2. Obiekt biblioteki jQuery

Pierwsza funkcja, która zostanie użyta w trakcie poznawania biblioteki jQuery, nosi nazwę `jQuery()`. Akceptuje ona maksymalnie dwa argumenty i zależnie od ich liczby i typu wykonuje różne zadania.

Podobnie do wielu innych (prawie wszystkich) metod w bibliotece funkcja ta pozwala na *tworzenie łańcucha*. Jest to technika programistyczna służąca do wywoływania kilku metod w jednej instrukcji. Zamiast używać kodu:

```
var obj = new Obj();  
obj.method();  
obj.anotherMethod();  
obj.yetAnotherMethod();
```

możesz napisać następujący kod:

```
var obj = new Obj();  
obj.method().anotherMethod().yetAnotherMethod();
```

Najczęstszym zastosowaniem funkcji `jQuery()` jest wybieranie elementów z modelu DOM, aby możliwe było wprowadzenie dla nich zmian. W tym przypadku funkcja

akceptuje dwa parametry: selektor i (opcjonalnie) kontekst. Funkcja zwraca obiekt zawierający kolekcję elementów modelu DOM spełniających podane kryteria. Czym jednak jest selektor?

Gdy w świecie technologii internetowych pojawiły się arkusze stylów CSS w celu oddzielenia części wizualnej od treści, niezbędny był sposób pozwalający na odwoływanie się do grup elementów strony z zewnętrznych arkuszy stylów. Opracowano metodę polegającą na wykorzystaniu selektorów, które w zwięzły sposób reprezentują elementy na podstawie ich typu, atrybutów lub położenia w obrębie dokumentu HTML. Osobom zaznajomionym z językiem XML może być znany język XPath (więcej informacji na jego temat dostępnych jest pod adresem <http://www.w3.org/TR/xpath20/>), który służy do wybierania elementów wewnątrz dokumentu HTML. Selektory CSS zapewniają równie duże możliwości, dostosowane są do użycia w obrębie stron HTML, cechują się trochę większą zwięzłością, a ponadto uważane są przeważnie za łatwiejsze do zrozumienia.

W bibliotece jQuery są stosowane te same selektory co w wypadku arkuszy stylów CSS. Biblioteka obsługuje nie tylko powszechnie implementowane selektory należące do standardu CSS 2.1, ale również bardziej zaawansowane selektory zdefiniowane w standardzie CSS 3. Jest to ważne, ponieważ część z tych selektorów może nie być w pełni implementowana przez wszystkie przeglądarki lub może nigdy się w nich nie pojawić (na przykład w starszych wersjach przeglądarki Internet Explorer). Jakby tego było mało, biblioteka jQuery oferuje też własne selektory i umożliwia tworzenie selektorów niestandardowych.

W wypadku tej książki będziesz mieć możliwość wykorzystania posiadanej wiedzy o arkuszach stylów CSS, aby móc szybko zacząć pracę. Później poznasz bardziej zaawansowane selektory obsługiwane przez bibliotekę jQuery. Jeśli nie masz zbyt dużej wiedzy na temat selektorów, nie przejmuj się. Bardziej szczegółowo selektory biblioteki jQuery zostaną omówione w rozdziale 2. Pełną listę selektorów możesz znaleźć w witrynie biblioteki jQuery pod adresem <http://api.jquery.com/category/selectors/>.

Załóżmy, że chcesz wybrać wszystkie znaczniki `<p>` na stronie za pomocą funkcji `jQuery()`. W tym celu możesz utworzyć następujący wiersz kodu:

```
var paragraphs = jQuery('p');
```

Biblioteka przeszukuje model DOM pod kątem pasujących elementów, począwszy od głównego elementu dokumentu. Oznacza to, że w wypadku ogromnej liczby elementów proces może być czasochłonny.

W większości przypadków możesz przyspieszyć wyszukiwanie za pomocą parametru `context`. Służy on do ograniczania procesu do jednego lub większej liczby poddrzew (zależnie od użytego selektora). Aby to wyjaśnić, zostanie zmodyfikowany poprzedni przykład.

Przyjmijmy, że zamierzasz znaleźć wszystkie znaczniki `<p>` zawarte w znaczniku `<div>`. Słowo „zawarte” nie oznacza, że znacznik `<div>` musi być nadrzędny względem znacznika `<p>`. Znacznik `<div>` może też być zwykłym elementem podrzędnym. Zadanie to możesz zrealizować za pomocą następującego kodu:

```
var paragraphsInDiv = jQuery('p', 'div');
```

Gdy użyjesz aliasu biblioteki jQuery, ta sama instrukcja będzie wyglądać następująco:

```
var paragraphsInDiv = $('p', 'div');
```

W przypadku zastosowania drugiego argumentu biblioteka jQuery zbiera najpierw elementy na podstawie tego selektora o nazwie *context*, a następnie pobiera elementy podrzędne pasujące do pierwszego parametru *selector*. Bardziej szczegółowo zagadnienie to zostanie omówione w rozdziale 2.

Jak wspomniano, funkcja `jQuery()` (oraz jej alias `$()`) zwraca obiekt języka JavaScript zawierający zestaw elementów modelu DOM zgodnych z selektorem w kolejności, w jakiej zdefiniowano je w dokumencie. Obiekt ten dysponuje dużą liczbą przydatnych, predefiniowanych metod, które mogą wykonywać działania względem zebranej grupy elementów. Terminy *kolekcja biblioteki jQuery*, *obiekt biblioteki jQuery* lub *zestaw biblioteki jQuery* (lub inne podobne określenia) będą używane w odniesieniu do zwracanego obiektu języka JavaScript zawierającego zestaw dopasowanych elementów, które mogą być przetwarzane za pomocą metod zdefiniowanych w bibliotece jQuery. Zgodnie z taką definicją podana wcześniej zmienna `paragraphsInDiv` to obiekt biblioteki jQuery zawierający wszystkie akapity, które są elementami podrzędnymi elementu `div`. Obiekty biblioteki jQuery będą intensywnie używane, gdy konieczne będzie wykonanie dla kilku elementów na stronie takich operacji, jak uruchamianie określonej animacji lub stosowanie stylu.

Jak wcześniej wspomniano, istotną cechą dużej liczby tych metod, które często są określane mianem *metod biblioteki jQuery*, jest to, że umożliwiają *tworzenie łańcucha*. Po zakończeniu działania metoda zwraca tę samą grupę elementów, jaką przetwarzała, po czym metoda gotowa jest na kolejne działanie. Gdy działania stopniowo stają się bardziej złożone, użycie oferowanej przez bibliotekę jQuery możliwości tworzenia łańcucha w dalszym ciągu będzie powodować zmniejszenie liczby wierszy kodu niezbędnego do uzyskania żądanych wyników.

Wcześniej w rozdziale zwrócono uwagę na korzyści wynikające z umieszczenia kodu JavaScript na dole strony. Od wielu lat projektanci umiejscawiali elementy script w znaczniku `<head>` strony, bazując na metodzie biblioteki jQuery o nazwie `ready()`. Takie rozwiązanie nie jest już zalecane, ale wielu projektantów nadal z niego korzysta. W następnym punkcie dowiesz się więcej na ten temat, a także poznasz rozwiązanie proponowane obecnie.

1.5.3. Procedura obsługi zdarzenia gotowości dokumentu

W przypadku zastosowania techniki przejrzystego kodu JavaScript zachowanie oddzielane jest od struktury. Użycie tego rozwiązania oznacza wykonywanie operacji dla elementów strony poza obrębem tworzących je znaczników dokumentu. Aby to osiągnąć, niezbędny jest sposób pozwalający na poczekanie przed wykonaniem tych operacji na pełną dostępność elementów modelu DOM strony.

W przykładzie grupy przycisków opcji cała treść musi zostać załadowana przed uzyskaniem możliwości zastosowania informacji o zachowaniu. W tym celu tradycyjnie

używana jest procedura obsługi zdarzenia `onload` instancji obiektu `window`, która wykonuje instrukcje po pełnym załadowaniu całej strony. Składnia kodu ma zwykle następującą postać:

```
window.onload = function() {  
    // działania do wykonania  
};
```

Kod ten powoduje, że zdefiniowany kod zostanie wykonany *po* całkowitym załadowaniu dokumentu. Niestety, przeglądarka nie tylko opóźnia wykonanie kodu procedury obsługi zdarzenia `onload` do momentu utworzenia drzewa modelu DOM, ale też oczekuje na pełne załadowanie wszystkich zasobów zewnętrznych i wyświetlenie strony w oknie przeglądarki. Obejmuje to takie zasoby jak obrazy, jak również pliki wideo QuickTime i Flash osadzone na stronach internetowych. W efekcie odwiedzający mogą doświadczyć poważnego opóźnienia między pierwszym wyświetleniem strony i wykonaniem skryptu procedury obsługi zdarzenia `onload`.

Co gorsza, jeśli załadowanie obrazu lub innego zasobu zajmuje znaczną ilość czasu, odwiedzający będą musieli poczekać na załadowanie obrazu, zanim staną się dostępne zaawansowane działania. Może to sprawić, że cała propozycja w postaci przejrzystego kodu JavaScript nie będzie mieć sensu w wypadku wielu rzeczywistych zastosowań.

Znacznie lepszym rozwiązaniem byłoby oczekiwanie przed wykonaniem skryptu stosującego zaawansowane działania tylko do momentu przeprowadzenia pełnej analizy struktury dokumentu i przekształcenia przez przeglądarkę kodu HTML w jego wynikowe drzewo modelu DOM. Osiągnięcie tego w sposób zapewniający zgodność z wieloma przeglądarkami, w tym i starszymi, jest w pewnym stopniu utrudnione. Biblioteka jQuery zapewnia jednak proste metody pozwalające spowodować wykonywanie kodu po załadowaniu drzewa modelu DOM (bez oczekiwania na zasoby zewnętrzne).

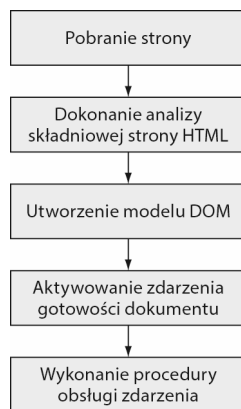
Składnia formalna służąca do definiowania odpowiedniego kodu ma następującą postać:

```
jQuery(document).ready(function() {  
    // W tym miejscu znajduje się napisany kod...  
});
```

Za pomocą funkcji `jQuery()` opakowywany jest najpierw obiekt `document`, a następnie wywoływana jest metoda `ready()`, której przekazywana jest funkcja do wykonania, gdy dokument będzie gotowy do zmodyfikowania. Oznacza to, że wewnątrz funkcji przekazanej metodzie `ready()` możesz bezpiecznie uzyskiwać dostęp do wszystkich elementów strony. Na rysunku 1.5 przedstawiono schemat opisanego mechanizmu.

Nie bez powodu zostało to nazwane *składnią formalną*. Postać skrócona jest następująca:

```
jQuery(function() {  
    // W tym miejscu znajduje się napisany kod...  
});
```



Rysunek 1.5. Reprezentacja graficzna kroków wykonywanych przez przeglądarkę przed wykonaniem procedury obsługi zdarzenia gotowości dokumentu

Przekazując funkcję funkcji `jQuery()` lub jej aliasowi `$()`, nakazujesz przeglądarce przed wykonaniem kodu poczekać do momentu pełnego załadowania modelu DOM (niczego ponadto). Jeszcze lepsze jest to, że techniki tej możesz użyć wiele razy w obrębie tego samego dokumentu HTML, a przeglądarka wykona wszystkie określone funkcje w takiej kolejności, w jakiej zadeklarowano je na stronie.

Dla porównania: technika oparta na zdarzeniu `onload` obiektu `window` zezwala na tylko jedną funkcję. Ograniczenie takie może też powodować generowanie trudnych do znalezienia błędów, jeśli w dowolnym dołączonym kodzie zewnętrznym na jego własne potrzeby używany jest mechanizm oparty na zdarzeniu `onload` (nie jest to najlepsze rozwiązanie).

Zastosowanie procedury obsługi zdarzenia gotowości dokumentu to dobry sposób wykorzystania techniki przejrzystego kodu JavaScript, ale użycie jej nie jest obowiązkowe, dlatego można z niej zrezygnować.

Ponieważ metoda `ready()` odpowiada za wykonanie kodu po załadowaniu modelu DOM, projektanci umieszczali elementy `script` w znaczniku `<head>` strony. Jak wspomniano w punkcie 1.2.2, „Oddzielanie skryptu”, elementy te możesz wstawić tuż przed znacznikiem zamykającym `</body>`. Dzięki temu możesz całkowicie uniknąć stosowania wywołania `$(document).ready()`, gdyż na tym etapie wszystkie inne elementy znajdują się już w modelu DOM. A zatem elementy te możesz bezpiecznie pobierać i ich używać. Aby zaznaczyć się z przykładem pokazującym, jak można uniknąć stosowania wywołania `$(document).ready()`, przyjrzyj się kodowi źródłowemu z pliku *rozdzial_1/radio.group.html*.

W pozostałej części książki będziemy się trzymać najlepszych praktyk, jakie obecnie obowiązują, dlatego nie będzie używana metoda `ready()`.

1.6. Podsumowanie

W niniejszym szybkim wprowadzeniu do biblioteki jQuery omówiono sporą ilość materiału. Podsumowując, biblioteka okazuje się przeważnie przydatna w przypadku dowolnej strony, która wymaga wykonania jakichkolwiek operacji dotyczących kodu JavaScript, z wyjątkiem tych najbardziej trywialnych. Szczególnie mocno skupiono się również na kwestii umożliwienia twórcom stron wykorzystania w ich obrębie zagadnienia przejrzystego kodu JavaScript. Dzięki tej technice zachowanie jest oddzielane od struktury w taki sam sposób, w jaki arkusze stylów CSS separują styl i strukturę. Zapewnia to lepszą organizację stron i zwiększoną wszechstronność kodu.

Pomimo że biblioteka jQuery wprowadza w przestrzeni nazw języka JavaScript tylko dwie nowe nazwy (funkcja o nazwie `jQuery` oraz jej alias `$`), zapewnia sporą porcję funkcjonalności. Jest to możliwe dzięki wszechstronności funkcji `jQuery`, która na podstawie przekazanych jej parametrów dostosowuje operację, jaką wykonuje.

Wspomniano, jak dobrze zorganizowane jest repozytorium biblioteki oraz kod w ogólności. Zwróciliśmy też szczególną uwagę na kilka dostępnych wersji biblioteki oraz różnice między nimi, aby umożliwić dokonanie świadomego wyboru. Ponieważ wydajność to istotna kwestia do uwzględnienia, zaprezentowaliśmy dostępne możliwości

zmniejszenia do minimum dodatkowego obciążenia wynikającego z dołączania biblioteki do stron. Użycie sieci CDN i dostosowywanie żądanych modułów to znakomity sposób na przyspieszenie operacji pobierania biblioteki jQuery.

W kolejnych rozdziałach zostaną objaśnione wszystkie funkcje, które biblioteka jQuery ma do zaoferowania projektantowi aplikacji internetowych. Nasze omówienie zaczniemy w następnym rozdziale, wyjaśniając, jak za pomocą selektorów biblioteki jQuery w szybki i prosty sposób identyfikować elementy, dla których mają zostać wykonane działania.

Skorowidz

A

- adaptowanie dodatków, 478
- Ajax, 309, 315
- aktualizowanie pakietów, 481, 483
- alias \$, 378
- analizowanie funkcji, 299
- animacja, 233, 260
 - efektu opadania, 257
 - efektu rozpraszania, 258
 - skalowania, 256
- animacje
 - niestandardowe, 253
 - wykonywane
 - jednocześnie, 260
- animowanie stanu
 - wyświetlania elementów, 238
- asercje, 444, 446
- atrybut, 116, 119
 - placeholder, 123

B

- BDD, 442
- biblioteka
 - jQuery, 31
 - Modernizr, 275
 - RequireJS, 475, 477
- błąd, 353

C

- CDN, Content Delivery Network, 40
- CLI, Command Line Interface, 44, 89, 217
- cudzysłowy, 65

D

- data, 405
- definicja AMD, 477
- definiowanie
 - atrybutów, 116
 - końców znaczników, 394
 - procedur obsługi zdarzeń, 184
 - właściwości, 116
- deklaracje funkcji, 509, 511
- dodatek, plugin, 52, 251, 369
 - Chosen, 375
 - isotope, 375
 - jCarousel, 375
 - jQuery Context Menu, 386, 390, 460
 - jQuery Photomatic, 394, 401
 - jQuery.deserialize, 320
 - Magnific Popup, 375
 - pickadate.js, 375
 - Photomatic, 395
 - typeahead.js, 375
 - Velocity, 375
- dodawanie
 - filtrów, 222
 - funkcji do kolejki, 262
 - nazw klas, 138
 - szablonów kontrolek, 225
- DOM, 46, 55, 90
- domknięcia, 516, 518
- dopasowywanie zestawu, 101
- dostęp do ustawień
 - domyślnych, 389
- dostępność, 363
- duże
 - projekty, 465
 - zestawy danych, 217
- dynamiczne ładowanie skryptu, 330

- dysk DVD, 216
- działania semantyczne, 182

E

- efekt, 233, 243, 362
 - opadania, 257
 - przełączanie, 362
 - rozpraszania, 258
- element
 - img, 87
- elementy
 - formularza, 170
 - nadrzędne, 81
 - zestawu, 91

F

- faza przechwytywania, 186
- filtr, 68
 - visible, 76
- filtrowanie
 - tablic, 283
 - zestawów danych, 217
- filtry
 - elementów podrzędnych, 69
 - formularza, 73
 - niestandardowe, 77
 - położenia, 68, 69
 - treści, 74
- flaga
 - noglobals, 455
 - notrycatch, 456
- folder dist, 44
- format
 - JSON, 329
 - JSONP, 343
- formatowanie daty, 404
- formularz, 170, 229
 - kontaktowy, 351

funkcja

getElementById, 501
 getElementsByName, 501
 \$.ajax, 339, 347
 \$.ajaxPrefilter, 348
 \$.ajaxSetup, 343
 \$.ajaxTransport, 349
 \$.contains, 303
 \$.each, 282
 \$.error, 307
 \$.extend, 291, 308
 \$.formatDate, 404
 \$.get, 328
 \$.getJSON, 330
 \$.getScript, 330
 \$.globalEval, 307
 \$.grep, 284
 \$.inArray, 287
 \$.isArray, 297
 \$.isEmptyObject, 297
 \$.isFunction, 297
 \$.isNumeric, 297
 \$.isPlainObject, 297
 \$.isWindow, 297
 \$.isXMLDoc, 297
 \$.makeArray, 287
 \$.map, 285
 \$.merge, 289
 \$.noConflict, 276, 279
 \$.noop, 302
 \$.param, 293
 \$.parseHTML, 301, 308
 \$.parseJSON, 300
 \$.parseXML, 300
 \$.post, 332
 \$.proxy, 304, 305
 \$.trim, 280
 \$.type, 299
 \$.unique, 288
 success, 422
 związania, 236

funkcje, 509, 511

asynchroniczne, 453
 jako wywołania zwrotne, 511, 513
 narzędziowe, 271, 297, 302, 403
 narzędziowe Ajax, 348
 sposobu animacji, 252

G

generowanie elementów, 86
 Git, 42
 Grunt, 42
 grupowanie testów, 456
 gwiazdka, 57

H

hierarchia
 elementów, 62
 obiektów, 506, 508

I

identyfikator
 \$, 277
 filters, 223
 IIFE, 518, 520
 implementacja
 dodatku Jqia Photomatic, 401
 kodu serwerowego, 355
 informacje o technologii Ajax, 310
 inicjowanie ządania, 313
 inspekcja instancji obiektu, 200
 instalowanie
 biblioteki jQuery, 37
 pakietów, 481
 środowiska Backbone.js, 493
 instancja obiektu event, 178
 interfejs wiersza poleceń, 44
 iteracja
 kolekcji, 281
 właściwości, 281

J

JavaScript, 34, 175
 jednorazowe nasłuchiwanie zdarzenia, 197
 język JavaScript, 175
 jQuery
 Compat, 38
 Mobile, 51
 UI, 51

K

kaskadowe elementy rozwijane, 333
 klasa
 animated-elements, 256
 found-element, 55
 template-title, 220
 kod
 asynchroniczny, 453
 HTML, 491
 JavaScript, 34
 serwerowy projektu, 355
 strony sklepu, 337
 synchroniczny, 445
 znaczników, 221, 354, 394
 kolejki efektów, 268
 kolejkovanie, 260
 animacji, 262
 funkcji, 262, 264
 kolekcja, 89, 487
 biblioteki, 47
 zadań do wykonania, 496
 komunikacja z serwerem, 309
 komunikat o błędzie, 352
 konfiguracja środowiska
 QUnit, 443, 458
 kontekst, 46, 80
 funkcji, 520
 kontrolka, 225
 konwencje nazewnicze, 376
 kursor myszy, 208

L

listy parametrów, 378
 literały
 obiektowe, 472, 507, 509
 tablicowe, 508, 510
 lokalizator dysków DVD, 215

Ł

ładowanie
 fragmentów dynamicznego kodu, 321
 modułów, 475
 skryptu, 330
 zawartości, 315, 317
 łańcuchy, 280

M

mechanizm przenoszenia
deklaracji, 510, 512

menedżer

pakietów, 42
zadań, 490

metoda

\$.Deferred, 415
\$.when, 421
abort, 311
add, 102
addBack, 112
addClass, 140
addEventListener, 186
after, 158
ajaxComplete, 350
ajaxError, 350
ajaxSend, 350
ajaxStart, 350
ajaxStop, 350
ajaxSuccess, 350
animate, 254
append, 155
appendTo, 160
async, 453
attr, 120, 121
before, 157
children, 98
clearQueue, 267
clone, 167
contents, 98
css, 144, 145
data, 129, 130, 131
deepEqual, 450
deferred.always, 433
deferred.done, 417
deferred.fail, 418
deferred.notify, 423
deferred.notifyWith, 423
deferred.progress, 424, 428
deferred.promise, 427
deferred.reject, 416
deferred.rejectWith, 417
deferred.resolve, 416
deferred.resolveWith, 416
deferred.state, 434
deferred.then, 430
define, 476
delay, 267
dequeue, 264

destroy, 388
detach, 166
each, 109, 256
empty, 166
end, 111
eq, 93
equal, 447
expect, 446
fadeIn, 246
fadeOut, 246
fadeTo, 247
finish, 250
first, 93
get, 92, 327
getAllResponseHeaders,
311
getElementsByTagName,
181, 188
getResponseHeader, 312
has, 108
hasClass, 143
height, 147
hide, 239, 269
hover, 210
html, 153
index, 95
init, 385
innerHeight, 150
innerWidth, 150
insertAfter, 160
insertBefore, 160
is, 110
isDefaultPrevented, 202
isImmediatePropagation
↳ Stopped, 202
isPropagationStopped, 202
jqiaPhotomatic, 394
jQuery.hasData, 134
last, 93
load, 317, 323
next, 98
nextAll, 98
nextUntil, 98, 99
not, 105
notDeepEqual, 450
notPropEqual, 450
notStrictEqual, 449
off, 198
offset, 151
offsetParent, 99
ok, 450

on, 191
one, 197
open, 312
outerHeight, 150
outerWidth, 150
overrideMimeType, 312
parents, 99
parentsUntil, 99
position, 152
prepend, 157
prependTo, 160
prev, 99
prevAll, 99
preventDefault, 182, 202
prevUntil, 99
promise, 434
prop, 125, 126, 127
queue, 263
QUnit.module, 457
QUnit.test, 445
remove, 165
removeAttr, 122
removeClass, 140
removeData, 133
removeProp, 127
replaceAll, 170
replaceWith, 169
scrollLeft, 152
scrollTop, 152
send, 312
serialize, 319
serializeArray, 320
setRequestHeader, 312
show, 242, 269
siblings, 100
slice, 107
slideDown, 248
slideToggle, 249
slideUp, 249
stop, 249
stopImmediatePropagation,
202
stopPropagation, 202
strictEqual, 448
text, 154, 155
then, 411, 429
throws, 451
toArray, 94
toggle, 238, 242
toggleClass, 141
trigger, 202, 204

metoda

- triggerHandler, 204
- unwrap, 163
- val, 171, 172
- validate, 494
- when, 420
- width, 146
- wrap, 162
- wrapAll, 163
- wrapInner, 163

metody

- asercji, 450
- biblioteki, 47
- obiektu Deferred, 415
- obiektu XMLHttpRequest, 311
- skrótowe, 206

model

- DOM, 46, 55, 90
- przeglądarki Internet Explorer, 189
- zadania do wykonania, 495
- zdarzeń, 175
- zdarzeń biblioteki jQuery, 189
- zdarzeń poziomu DOM, 183
- zdarzeń przeglądarek, 174

moduły, 471

- panelu sterowania, 236

modyfikacja

- elementów, 54
- kodu dodatku, 383
- kolekcji, 279
- obiektów, 279
- właściwości elementu, 125

N

- narzędzia dla programistów, 59, 327

narzędzie

- Bower, 479
- Git, 42
- Grunt, 42
- Node.js, 43
- npm, 42

- nasłuchiwanie zdarzeń, 197
- nazwy funkcji, 509, 511
- nowe elementy HTML, 86

O

- obcinanie łańcuchów, 280
- obiekt, 504, 506
 - biblioteki, 45
 - Deferred, 409, 414, 424, 433
 - Event, 178, 200
 - nasłuchujący zdarzenia, 174
 - Promise, 414, 426
 - QUnit.config, 458
 - window, 508, 510
 - XHR, 310
 - XMLHttpRequest, 311
- obiekty pierwszoklasowe, 509, 511
- obietnice, 410
- obserwowanie selektora, 55
- obsługa
 - stanu gotowości, 314
 - zdarzeń, 47, 176, 184, 190, 360, 433
 - zdarzeń Ajax, 344, 346
- oddzielanie skryptu, 36
- odrzućanie obiektu Deferred, 415
- okno dodatku Photomatic, 395
- określanie wartości wyrażeń, 306
- opakowywanie elementów, 161, 164
- opcje
 - funkcji \$.ajax, 339, 340, 342
 - wyboru, 455
- operator new, 509
- opóźnianie kolejkowanych funkcji, 267
- optymalizowanie filtrów, 469
- organizowanie kodu, 471

P

- pakiet testów, 459
- panel selektora, 54
- parametr
 - context, 468
 - easing, 251
 - options, 241

PHP, 355

- platforma Node.js, 42
- plugins, 52, 251, 369
- pobieranie elementów, 62
- podzestaw, 107
- pokaz slajdów, 392
- powiadamianie o postępie procesu, 423
- powielanie elementów, 167
- poziom DOM, 175
 - Level 0, 175
 - Level 2, 183
- procedura obsługi zdarzeń, 47, 177, 181, 360, 433
- projektowanie BDD, 442
- propagacja zdarzeń, 179, 182, 186
- propEqual, 450
- propozycja
 - Promises/A, 413
 - Promises/A+, 411
- przeglądarka, 39
 - Internet Explorer, 189
- przełączanie efektów, 362
- przenoszenie elementów, 155, 160
- przestrzeń nazw, 381, 384
 - dla zdarzeń, 211
- przesuwanie elementów, 248
- przetwarzanie kolekcji, 85
- pseudoklasa
 - nth-child(), 70

Q

- QUnit, 51

R

- relacje, 96
- replikacja szablonu, 219
- router, 489
 - w Backbone.js, 490
- rozstrzyganie obiektu Deferred, 415
- rozszerzanie
 - biblioteki, 370
 - obiektów, 290

S

selektor, 46
 nth-child(), 70
 nth-last-child(), 70
 nth-last-of-type(), 70
 selektor
 atrybutów, 64
 CSS, 57, 64
 elementów podrzędnych, 61, 63, 81
 identyfikatora, 60
 klasy, 61, 468
 niestandardowy, 74
 uniwersalny, 57, 467
 serializacja
 danych formularza, 319
 wartości parametrów, 292
 zagnieżdżonych parametrów, 294
 sieć CDN, 40
 silnik selektorów Sizzle, 471
 skalowanie, 256
 składniki, 51
 słowo kluczowe
 function, 509, 511
 this, 304, 512, 514
 specyfikacja WAI-ARIA, 364
 sposób animacji, 240
 sprawdzanie
 pola, 356
 poprawności danych, 353, 358
 poprawności pól, 357
 przynależności, 303
 wyników, 331
 stan obiektu Deferred, 433
 stopniowe
 rozjaśnianie, 245
 wyświetlanie, 239
 stosowanie obietnic, 434
 struktura, 35
 biblioteki jQuery, 42
 obiektu JSON, 353
 style elementów, 138
 system kontroli wersji, 42
 szablony kontrolki, 225
 szybkość animacji, 273

Ś

ściemnianie elementów, 245
 śledzenie
 postępu, 424
 postępu działań, 314
 propagacji zdarzenia, 186
 środowiska testowania
 jednostkowego, 441
 środowisko
 Backbone.js, 483, 485, 490, 493
 MV*, 485
 QUnit, 442, 445, 455

T

tablice, 94
 filtrowanie, 283
 języka JavaScript, 286
 translacja, 285
 technika TDD, 440
 technologia Ajax, 309, 359
 test wydajności, 469
 testowanie, 298, 438
 jednostkowe, 437, 439
 obiektów, 296
 za pomocą asercji, 446
 zadań asynchronicznych, 453
 translacja tablic, 285
 tworzenie
 animacji
 niestandardowych, 253
 dodatku, 375
 dodatku Jqia Photomatic, 396
 elementów, 219
 filtrów niestandardowych, 77
 funkcji formatującej datę, 404
 instancji obiektu XHR, 310
 kodu HTML, 491
 kodu znaczników, 221, 354
 łańcucha, 45
 łańcucha metod, 389
 menedżera zadań, 490

niestandardowych funkcji narzędziowych, 403
 pokazu slajdów, 392
 projektu, 352
 strony, 337
 testów, 445
 własnej wersji, 43
 zdarzeń
 niestandardowych, 210
 zestawu, 91
 żądań Ajax, 338
 żądań GET i POST, 325, 332
 typ danych Object, 504, 506
 typy filtrów, 230

U

ukrywanie
 elementów, 234
 okna dialogowego, 361
 ulepszenie progresywne, 364
 usługa jsPerf, 469
 ustawianie
 stylów, 143
 wartości domyślnych
 żądania, 342
 wymiarów, 146
 usuwanie
 atrybutów, 122
 elementów, 165
 filtrów, 227
 nazw klas, 138
 pakietów, 481, 483
 procedur obsługi zdarzeń, 198
 uzupełniacz, 275
 uzyskiwanie
 danych, 327, 329
 odpowiedzi, 314
 stylów, 143
 zestawów, 96
 użycie
 aliasu \$, 378
 atrybutów, 115, 119
 biblioteki RequireJS, 477
 danych, 115
 dodatku, 371, 373
 metody deferred.progress, 424, 428

użycie
 obiektu Promise, 418, 421, 426
 selektora uniwersalnego, 59
 właściwości, 115, 272
 zestawu, 110

W

wartości
 atrybutów, 119, 120
 domyślne żądania, 342
 elementów formularza, 170
 kontekstu funkcji, 514, 516
 wersja, 37
 bez kompresji, 40
 zminimalizowana, 40
 wiązanie
 konkretnego zdarzenia, 207
 kontekstów funkcji, 304
 widok, 487
 aplikacji, 498
 zadania do wykonania, 497
 właściwości, 116
 biblioteki, 272
 elementu, 125
 obiektu, 505, 507
 obiektu jQuery.Event, 201
 obiektu window, 508, 510
 właściwość
 \$.browser, 273
 \$.support, 274
 fx.off, 45
 whatAmI, 513, 515
 wstawianie funkcji do kolejek, 268
 wybieranie
 elementów, 64
 wersji, 37
 wydajność, 40, 80
 selektorów, 467
 wyjątek, 307
 wykrywanie typu wartości, 298
 wyłączanie animacji, 273
 wyrażenia funkcji, 509, 511
 wyszukiwanie pakietu, 481

wyświetlanie
 elementów, 234, 238
 stopniowe, 239
 wymiarów elementu, 149
 wyników, 227
 wywołania zwrotne, 409, 511, 513
 wyzwalanie
 konkretnego zdarzenia, 207
 procedur obsługi zdarzeń, 201
 zdarzeń, 205
 wzorzec
 IIFE, 518, 520
 MVC, 486
 oparty na literałach obiektyowych, 472
 oparty na modułach, 473
 projektowy IIFE, 276

Z

zachowanie, 35
 zaciemnianie kodu, 40
 zadanie do wykonania, 494
 zależności, 479
 zarządzanie
 kolekcją, 89
 zależnościami, 479
 zasięg deklaracji funkcji, 516, 518
 zastępowanie
 elementów, 168
 kodu HTML, 153
 zastosowanie
 metody promise, 435
 zdarzeń, 216
 zatrzymywanie animacji, 249
 zawartość elementu, 153
 zdarzenia, 47, 173, 180
 Ajax, 345
 biblioteki jQuery, 189
 globalne, 344
 lokalne, 344
 przeglądarek, 174
 zestaw testów, 96, 460
 dopasowywanie, 101
 iteracja elementów, 109
 określanie wielkości, 91

przekształcanie
 elementów, 108
 udoskonalanie zawartości, 104
 uzyskiwanie elementów, 91
 uzyskiwanie podzestawów, 107
 zgłaszanie wyjątków, 307
 zmienianie
 stylów elementów, 138
 szybkości animacji, 273
 znacznik
 <div>, 46
 <p>, 46
 <script>, 181
 , 492
 znajdowanie
 dodatków, 371
 indeksu elementu, 94
 zwiększanie wydajności, 40, 80
 selektorów, 467
 zwijanie, 235, 236

Ż

żądania Ajax, 338, 419
 żądanie
 GET, 325
 POST, 325

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA



Helion SA

Lekka biblioteka jQuery znacząco ułatwia kodowanie w JavaScriptcie. Dzięki niej programiści nie muszą już ręcznie zarządzać obsługą selektorów CSS czy nawigacją w modelu DOM, a praca nad animacjami jest o wiele łatwiejsza. Co ważne, projekt ten wciąż się rozwija: w bibliotece jQuery 3 dodano kolejne funkcje, które sprawiły, że praca projektanta aplikacji internetowych stała się bezproblemowa i efektywna. Okazuje się, że wobec rosnącego znaczenia JavaScriptu w tworzeniu aplikacji internetowych umiejętność postępowania się jQuery to bardzo ważny atut na rynku pracy.

Jeśli znasz choćby podstawy JavaScriptu i chciałbyś możliwie szybko nabrać biegłości w postępowaniu się jQuery, trzymasz w dłoni właściwą książkę. Dzięki temu zwięztemu przewodnikowi nauczysz się płynnie realizować zadania, które pojawiają się niemal zawsze podczas tworzenia aplikacji internetowych. Dowiesz się, jak nawigować w obrębie modelu DOM, obsługiwać zdarzenia, tworzyć dodatki do jQuery i żądania Ajax, a nawet przeprowadzać testy jednostkowe kodu. Oczywiście nie zabrakło tu rzeczywistych przykładów kodu, ilustrujących każde omawiane zagadnienie. Ponadto niniejsze wydanie rozbudowano o rozdziały, w których omówiono współdziałanie jQuery z innymi narzędziami i środowiskami, a także budowę nowoczesnych aplikacji z jedną stroną w środowisku Backbone.js.

Dzięki tej książce poznasz:

- budowę, przeznaczenie i zasady funkcjonowania biblioteki jQuery
- nowe praktyczne możliwości biblioteki jQuery 3
- model DOM i obsługę zdarzeń
- sposoby tworzenia animacji i inne efekty związane z szatą graficzną aplikacji
- technologię Ajax i podstawy jej obsługi
- sposoby na testy jednostkowe i poprawę wydajności kodu

Bear Bibeault

programuje od ponad trzydziestu lat. Uzyskał dwa stopnie naukowe z dziedziny elektrotechniki, jest też współautorem licznych książek dotyczących programowania.

Yehuda Katz

od wielu lat angażuje się w projekty związane z oprogramowaniem *open source*. Współtworzył bibliotekę jQuery i środowisko Ember.js.

Aurelio De Rosa

jest doświadczonym projektantem aplikacji internetowych i członkiem zespołu rozwijającego bibliotekę jQuery. Tworzy oprogramowanie Internetowe z wykorzystaniem stosu WAMP oraz języków: HTML5, CSS3, Sass, JavaScript i PHP.

Poznaj jQuery, korzystaj z tej biblioteki i ciesz się nią!

Helion

44493

numer katalogowy

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:

🔗 <http://helion.pl/promocje>

Książki najchętniej czytane:

🔗 <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

🔗 <http://helion.pl/novosci>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-283-2275-2



9 788328 322752

Informatyka w najlepszym wydaniu

cena: 89,00 zł