

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

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

Java Script. Biblia

Autor: Danny Goodman

Tłumaczenie: Grzegorz Kowalski, Piotr Rajca

ISBN: 83-7197-694-1

Tytuł oryginału: [JavaScript Bible 4th edition](#)

Format: B5, stron: 1460



JavaScript to jeden ze składników DHTML, umożliwiający tworzenie efektów niedostępnych w standardowym HTML-u, dzięki którym Twoja strona WWW stanie się dynamiczna i interaktywna.

Istnieje wiele książek na temat JavaScriptu. Język ten od wielu lat wykorzystywany jest przez twórców stron WWW stając się niezbędnym, a wręcz podstawowym narzędziem, które opanować powinien każdy webmaster i programista. JavaScript zyskuje coraz większą popularność, a jego implementacje, obsługiwane przez nowe generacje przeglądarek są coraz bogatsze w funkcje.

„JavaScript. Biblia” to książka szczególna. Znajdziesz w niej po prostu wszystko, co wiadomo o JavaScriptcie. Co więcej, informacje te przekazuje Danny Goodman, autor licznych bestsellerów, autorytet w dziedzinie języków skryptowych i doskonały nauczyciel. Znany jest także ze swych artykułów „JavaScript Apostle” publikowanych w internetowym magazynie informacyjnym „ViewSource” firmy Netscape. Danny Goodman gwarantuje rzetelność informacji zawartych w książce.

Znajdziesz w niej:

- Kompletny, szczegółowy i przystępny opis języka JavaScript
- Omówienie modeli dokumentów implementowanych w różnych przeglądarkach
- Opis wszystkich funkcji języka JavaScript z przykładami ich zastosowania
- Omówienie kilku zaawansowanych aplikacji napisanych w JavaScriptcie, zawierające wiele cennych wskazówek dla programistów.

Wydanie, które trzymasz w ręku, zostało zaktualizowane i obejmuje funkcje JavaScriptu dostępne w Netscape 6.0. Niezależnie od tego, czy dopiero zaczynasz przygodę z projektowaniem stron WWW, czy też jesteś doświadczonym webmasterem, jest to książka dla Ciebie. To jedyna książka o JavaScriptcie jakiej potrzebujesz, żeby poznać i wykorzystać go w pełni.



Spis treści

Przedmowa	25
O Autorze	28
Wstęp	29
Część I Pierwsze kroki w JavaScriptcie	37
Rozdział 1. Rola języka JavaScript w sieci WWW i poza nią	39
Konkurencja w sieci WWW	40
Język HTML	41
Skrypty CGI.....	41
Programy pomocnicze i moduły rozszerzające	42
Aplety Javy	43
JavaScript — język dla każdego	44
LiveScript staje się JavaScriptem	45
Świat Microsoftu	46
JavaScript — właściwe narzędzie do odpowiednich zadań	46
Rozdział 2. Wyzwania dla twórców w świecie walki przeglądarek	49
Gra w przeskakiwanie	50
Uniki i ochrona	51
Kwestie zgodności w chwili obecnej.....	51
Oddzielanie języka od obiektów	52
Standard rdzennej części języka	52
Obiektowy model dokumentu	54
Kaskadowe arkusze stylów	55
Dynamiczny HTML	55
Rozwijanie strategii pisania skryptów	56
Rozdział 3. Twój pierwszy skrypt w języku JavaScript	57
Narzędzia programistyczne	57
Wybór edytora tekstów	57
Wybór przeglądarki	58
Przygotowanie środowiska pracy	58
Windows.....	59
MacOS.....	60
Kwestie przeladowywania strony.....	61
Zadanie twojego pierwszego skryptu	61

Stworzenie pierwszego skryptu	62
Analiza treści skryptu	63
Znacznik <SCRIPT>	63
Skrypt dla każdej przeglądarki	64
Wyświetlanie tekstu	65
Pobaw się	66
Część II Kurs JavaScriptu	67
Rozdział 4. Przeglądarka i obiekty dokumentu	69
Skrypty umożliwiają tworzenie aktywnych dokumentów	69
JavaScript w akcji	70
Interaktywne interfejsy użytkownika	71
Proste wyszukiwanie danych	72
Zatwierdzanie poprawności formularzy	72
Interaktywne dane	73
Obsługa wielu ramek	74
Dynamiczny HTML	75
Kiedy używać języka JavaScript	75
Obiektowy model dokumentu	76
Hierarchia zawierania	78
Kiedy dokument jest ładowany	79
Prosty dokument	79
Dodanie formularza	80
Dodanie tekstowego pola wejściowego	80
Dodanie przycisku	81
Odniesienia do obiektów	81
Nazewnictwo obiektów	82
Prosty dokument	83
Dodanie formularza	83
Dodanie tekstowego pola wejściowego	84
Dodanie przycisku	84
O składni wykorzystującej znak kropki	84
Model organizacji grupy dyskusyjnej	85
Co definiuje obiekt	85
Właściwości	86
Metody	87
Procedury obsługi zdarzeń	88
Ćwiczenia	89
Rozdział 5. Skrypty i dokumenty HTML	91
Miejsce skryptu w dokumencie	91
Znacznik <SCRIPT>	91
Rozmieszczenie znaczników	92
Obsługa starszych przeglądarek	94
Instrukcje języka JavaScript	95
Czas wykonania instrukcji skryptu	96
Podczas ładowania dokumentu — natychmiastowe wykonanie	96
Odroczone wykonanie skryptu	97
Przeglądanie błędów w skrypcie	98

Pisanie skryptów a programowanie	100
Ćwiczenia	101
Rozdział 6. Podstawy programowania, część I	103
Co to za język?	103
Przetwarzanie informacji	104
Zmienne	104
Utworzenie zmiennej	105
Nazwy zmiennych	106
Obliczanie wartości wyrażeń	106
Wyrażenia w pliku script1.htm	107
Wyrażenia i zmienne	108
Przekształcanie typów danych	109
Konwersja ciągu znaków na liczbę	110
Konwersja liczby na ciąg znaków	110
Operatory	111
Operatory arytmetyczne	111
Operatory porównania	112
Ćwiczenia	112
Rozdział 7. Podstawy programowania, część II	115
Decyzje i pętle	115
Instrukcje sterujące	116
Instrukcja if	116
Instrukcja if...else	117
Pętle	118
Funkcje	119
Parametry funkcji	120
Zasięg zmiennej	121
Nawiasy klamrowe	123
Tablice	123
Tworzenie tablic	124
Dostęp do danych z tablicy	125
Równoległe tablice	125
Obiekty dokumentu przechowywane w tablicach	127
Ćwiczenia	128
Rozdział 8. Obiekty okna i dokumentu	129
Obiekty dokumentu	129
Obiekt window	129
Dostęp do metod i właściwości obiektu window	130
Tworzenie okna	131
Metody i właściwości obiektu window	133
Właściwość window.status	133
Metoda window.alert()	134
Metoda window.confirm()	134
Metoda window.prompt()	135
Procedura obsługi zdarzenia onLoad()	136
Obiekt location	136
Obiekt history	137

Obiekt document.....	137
Właściwość document.forms[].....	137
Właściwość document.title.....	138
Metoda document.write().....	138
Obiekt dowiązania.....	140
Ćwiczenia.....	141
Rozdział 9. Formularze i elementy formularzy.....	143
Obiekt FORM.....	143
Formularz jako obiekt i kontener.....	144
Dostęp do właściwości formularza.....	144
Właściwość form.elements[].....	144
Elementy formularza jako obiekty.....	145
Obiekty tekstowe.....	145
Obiekt button.....	147
Obiekt checkbox.....	148
Obiekt radio.....	149
Obiekt SELECT.....	150
Przekazywanie danych z formularza do funkcji.....	152
Przesyłanie i wstępne sprawdzanie poprawności formularzy.....	154
Ćwiczenia.....	156
Rozdział 10. Łańcuchy znaków, daty i operacje matematyczne.....	157
Obiekty z rdzennej części języka.....	157
Łańcuchy znaków to też obiekty.....	158
Łączenie ciągów znaków.....	158
Metody obiektu string.....	159
Obiekt math.....	161
Obiekt date.....	163
Obliczenia na datach.....	164
Ćwiczenia.....	166
Rozdział 11. Obsługa skryptowa ramek i wielu okien.....	167
Ramki: Rodzice i Dzieci.....	167
Odwołania między „członkami rodziny”.....	169
Odwołania rodziców do dzieci.....	169
Odwołania dzieci do rodziców.....	170
Odwołania między dziećmi.....	170
Porady odnośnie do obsługi ramek.....	171
Kontrola wielu ramek — paski nawigacyjne.....	171
Więcej o odwołaniach do obiektów klasy Window.....	174
Ćwiczenia.....	175
Rozdział 12. Obrazki oraz Dynamiczny HTML.....	177
Obiekt Image.....	177
Zamiana obrazków.....	178
Buforowanie obrazków w pamięci podręcznej przeglądarki.....	178
Zamienianie grafiki pod kursorem.....	180
Więcej dynamiki w HTML-u.....	184
Ćwiczenia.....	184

Część III Opis obiektów dokumentu 185

Rozdział 13. Podstawy JavaScriptu 187

Wersje języka JavaScript.....	187
Standard rdzennej części języka — ECMAScript.....	188
Osadzanie skryptów w dokumentach HTML.....	189
Znaczniki <SCRIPT>.....	189
Ukrywanie treści skryptu przed starszymi przeglądarkami.....	191
Czy można zupełnie ukryć skrypt?.....	192
Biblioteki skryptów (pliki .js).....	192
Kwestie zgodności bibliotek.....	194
Encje JavaScriptu w przeglądarkach Navigator 3 i 4.....	194
Wykrywanie wersji przeglądarki.....	195
Czy obsługa JavaScriptu jest uaktywniona?.....	195
Tworzenie skryptów dla wielu przeglądarek.....	197
Projektowanie aplikacji pod kątem zapewnienia zgodności.....	205
Przeglądarki w wersji beta.....	206
Aplikacja The Evaluator Sr.....	207
Tabele zgodności w rozdziałach poświęconych opisowi języka.....	208
Podstawy języka dla doświadczonych programistów.....	209
Dalej ku modelom obiektowym.....	212

Rozdział 14. Podstawy obiektowego modelu dokumentu..... 213

Hierarchia modelu obiektowego.....	213
Hierarchia jako mapa drogowa.....	214
Mapa drogowa obiektów dokumentu w przeglądarce.....	215
Jak rodzą się obiekty dokumentu.....	216
Właściwości obiektu.....	216
Metody obiektu.....	217
Obiektowe procedury obsługi zdarzeń.....	218
Procedury obsługi zdarzeń jako metody.....	218
Procedury obsługi zdarzeń jako właściwości.....	219
„Szwedzki stół” modeli obiektowych.....	220
Bazowy model obiektowy.....	221
Bazowy model obiektowy rozszerzony o obrazki.....	222
Rozszerzenia obecne tylko w przeglądarce Navigator 4.....	222
Model przechwytywania zdarzeń.....	223
Warstwy.....	223
Rozszerzenia w przeglądarkach Internet Explorer 4+.....	224
Elementy HTML-a jako obiekty.....	224
Hierarchia zawierania elementów.....	226
Kaskadowe arkusze stylów.....	227
Propagacja zdarzeń.....	227
Dowiązanie skryptów do obsługi zdarzeń.....	228
Funkcje Win32.....	228
Rozszerzenia w przeglądarkach Internet Explorer 5+.....	229
W3C DOM.....	230
Poziomy DOM.....	231
Co się nie zmienia.....	231
Co jest niedostępne.....	232
„Nowe” praktyki w HTML-u.....	232

Nowe koncepcje w modelu DOM	233
Statyczne obiekty języka HTML w modelu W3C DOM	242
Dwukierunkowy model zdarzeń	244
Mieszanie modeli obiektowych	245
Podejście konserwatywne	246
Podejście kompromisowe	247
Podejście radykalne	250
Obsługa zdarzeń	252
Symulacja składni IE 4+ w przeglądarce NN 6	252
Symulacja właściwości all	253
Symulacja właściwości treści	254
Dokąd skierować się z tego punktu	256
Rozdział 15. Obiekty ogólnych elementów HTML	257
Obiekty ogólne	258
Składnia	261
O obiektach	261
Właściwości	261
Metody	306
Procedury obsługi zdarzeń	352
Popularne sposoby obsługi klawiatury	371
Rozdział 16. Obiekty window oraz frame	381
Terminologia	382
Ramki	382
Tworzenie ramek	382
Model obiektów dokumentu zawierającego ramki	383
Odwołania do ramek	384
Różnice pomiędzy oknami top i parent	385
Zapobieganie wyświetlaniu dokumentu w ramce podrzędnej	385
Wymuszanie zastosowania ramek	386
Wyłączanie ramek	386
Dziedziczenie a struktura hierarchiczna	387
Synchronizacja ramek	387
Puste ramki	388
Przeglądanie kodu źródłowego ramki	389
Ramki a obiekty elementów FRAME	389
Obiekt window	390
Składnia	392
O obiekcie	392
Właściwości	394
Metody	422
Procedury obsługi zdarzeń	465
Obiekt elementu FRAME	472
Składnia	473
Kilka słów o obiekcie	473
Właściwości	473
Obiekt elementu FRAMESET	479
Składnia	479
Kilka słów o obiekcie	480
Właściwości	481

Obiekt elementu IFRAME.....	484
Składnia	485
Kilka słów o obiekcie	485
Właściwości	486
Obiekt popup	491
Składnia	491
Kilka słów o obiekcie	491
Właściwości	492
Metody	493
Rozdział 17. Obiekty location i history	495
Obiekt location	496
Składnia	496
Kilka słów o obiekcie	496
Właściwości	498
Metody	505
Obiekt history	507
Składnia	507
O obiekcie	507
Właściwości	509
Metody	510
Rozdział 18. Obiekty document oraz body	515
Obiekt document.....	516
Składnia	518
O obiekcie	518
Właściwości	520
Metody	558
Procedury obsługi zdarzeń	580
Obiekt BODY	581
Składnia	581
O obiekcie	581
Właściwości	582
Metody	587
Procedury obsługi zdarzeń	590
Rozdział 19. Obiekty elementów tekstowych	591
Obiekty elementów BLOCKQUOTE oraz Q.....	592
Składnia	592
O obiektach	592
Właściwości	592
Obiekt elementu BR	593
Składnia	593
O obiekcie	593
Właściwości	593
Obiekt elementu FONT	594
Składnia	594
O obiekcie	594
Właściwości	595
Obiekty elementów od H1 do H6.....	596
Składnia	597
O obiektach	597
Właściwości	597

Obiekt elementu HR	598
Składnia	598
O obiekcie	598
Właściwości	598
Obiekt elementu LABEL	601
Składnia	601
O obiekcie	601
Właściwości	601
Obiekt elementu MARQUEE	603
Składnia	603
O obiekcie	603
Właściwości	604
Metody	607
Procedury obsługi zdarzeń	608
Obiekt Range	609
Składnia	609
O obiekcie	610
Wykorzystanie zakresów tekstowych	612
Właściwości	613
Metody	616
Obiekt selection	627
Składnia	627
O obiekcie	627
Właściwości	629
Metody	629
Obiekty Text oraz TextNode	631
Składnia	631
O obiekcie	632
Właściwości	632
Metody	633
Obiekt TextRange	635
Składnia	636
O obiekcie	636
Operowanie na zakresach tekstowych	637
Informacje dotyczące zgodności przeglądarek	638
Właściwości	639
Metody	641
Obiekt TextRectangle	658
Składnia	658
O obiekcie	659
Właściwości	659
Rozdział 20. Obiekty dyrektyw języka HTML — podsumowanie	661
Obiekt dla elementu HTML	662
Składnia	662
O obiekcie	662
Właściwość	662
Obiekt dla elementu HEAD	663
Składnia	663
O obiekcie	663
Właściwość	663

Obiekt dla elementu BASE.....	664
Składnia	664
O obiekcie	664
Właściwości	664
Obiekt dla elementu BASEFONT	665
Składnia	666
O obiekcie	666
Właściwości	666
Obiekt dla elementu ISINDEX.....	667
Składnia	668
O obiekcie	668
Obiekt dla elementu LINK	668
Składnia	669
O obiekcie	669
Właściwości	669
Procedury obsługi zdarzeń	673
Obiekt dla elementu META	673
Składnia	673
O obiekcie	673
Właściwości	674
Obiekt dla elementu SCRIPT	676
Składnia	676
O obiekcie	676
Właściwości	677
Obiekt dla elementu TITLE.....	679
Składnia	679
O obiekcie	679
Właściwość.....	680
Rozdział 21. Obiekty łącza i kotwicy	681
Obiekty dla łącza, kotwicy i elementu A.....	682
Składnia	682
O obiekcie	683
Właściwości	686
Rozdział 22. Obiekty Image, Area oraz Map	693
Obrazy oraz obiekty elementu IMG	693
Składnia	694
O obiekcie	695
Właściwości	698
Procedury obsługi zdarzeń	710
Obiekt AREA	711
Składnia	712
O obiekcie	712
Właściwości	713
Obiekt MAP	714
Składnia	715
O obiekcie	715
Właściwość.....	715

Rozdział 23. Obiekt form oraz obiekty z nim związane	717
Miejsce formularza w hierarchii obiektów	718
Obiekt FORM	718
Składnia	719
O obiekcie	719
Odwołania do elementów sterujących formularzy	720
Przekazywanie formularzy i ich elementów do funkcji	720
Przesyłanie formularzy pocztą elektroniczną	724
Modyfikacja atrybutów formularzy	726
Przyciski w formularzach	726
Przekierowanie po przesłaniu formularza	727
Tablice elementów formularza	728
O obiektach elementu <INPUT>	729
Właściwości	729
Metody	736
Procedury obsługi zdarzeń	738
Obiekty elementów FIELDSET oraz LEGEND	739
Składnia	739
O obiektach	740
Obiekt elementu LABEL	741
Składnia	741
O obiekcie	741
Właściwości	742
Rozdział 24. Obiekt button	745
Obiekt elementu BUTTON oraz obiekty przycisków button, submit oraz reset	745
Składnia	746
O obiektach	746
Właściwości	748
Metody	751
Procedury obsługi zdarzeń	751
Obiekt checkbox	752
Składnia	753
O obiekcie	753
Właściwość	754
Metody	756
Procedury obsługi zdarzeń	757
Obiekt radio	757
Składnia	757
O obiekcie	758
Właściwości	759
Metody	762
Procedury obsługi zdarzeń	763
Obiekt przycisku typu image	763
Składnia	764
O obiekcie	764
Właściwości	764

Rozdział 25. Obiekty tekstowych elementów formularzy	767
Obiekt text (pole tekstowe).....	768
Składnia	768
O obiekcie	769
Pola tekstowe a zdarzenia	770
Wartości pól tekstowych a zachowywanie danych	772
Właściwości	773
Metody	778
Procedury obsługi zdarzeń	780
Obiekt password (pola hasła)	782
Składnia	782
O obiekcie	782
Obiekt hidden (pole ukryte).....	783
Składnia	783
O obiekcie	783
Obiekt elementu TEXTAREA.....	784
Składnia	784
O obiekcie	785
Znaki końca wiersza w wielowierszowych polach tekstowych	785
Właściwości	786
Metody	787
Rozdział 26. Obiekty select, option oraz FileUpload.....	789
Obiekt elementu SELECT	789
Składnia	790
O obiekcie	790
Modyfikacja opcji list (NN 3+, IE 4+).....	792
Modyfikacja opcji list (IE 4+).....	796
Modyfikacja list (DOM W3C).....	798
Właściwości	799
Metody	805
Procedury obsługi zdarzeń	806
Obiekt elementu OPTION.....	807
Składnia	808
O obiekcie	808
Właściwości	809
Obiekt elementu OPTGROUP.....	809
Składnia	809
O obiekcie	810
Właściwości	810
Obiekt elementu INPUT typu file	811
Składnia	811
O obiekcie	811
Rozdział 27. Obiekty tabel i list	813
Hierarchia obiektów związanych z tabelami	814
Określanie zawartości komórek tabel	816
Modyfikacja zawartości komórek tabel	817
Modyfikacja wierszy tabeli	821
Modyfikacja kolumn tabeli	826
Klasy obiektów DOM W3C związanych z tabelami.....	828

Obiekt elementu TABLE	829
Składnia	830
O obiekcie	830
Właściwości	831
Metody	840
Obiekty elementów TBODY, TFOOT oraz THEAD	844
Składnia	844
O obiektach	845
Właściwości	845
Obiekt elementu CAPTION	846
Składnia	847
O obiekcie	847
Obiekty elementów COL oraz COLGROUP	847
Składnia	848
O obiektach	848
Właściwości	849
Obiekt elementu TR	850
Składnia	850
O obiekcie	851
Właściwości	852
Metody	853
Obiekty elementów TD i TH	854
Składnia	855
O obiektach	856
Właściwości	856
Obiekt elementu OL	859
Składnia	859
O obiekcie	859
Właściwości	860
Obiekt elementu UL	862
Składnia	862
O obiekcie	862
Właściwości	863
Obiekt elementu LI	863
Składnia	863
O obiekcie	864
Właściwości	864
Obiekty elementów DL, DT oraz DD	865
Składnia	865
O obiektach	865
Obiekty elementów DIR i MENU	866
Składnia	866
O obiektach	866
Rozdział 28. Obiekt navigator oraz inne obiekty środowiskowe	867
Obiekty clientInformation (IE 4+) oraz navigator(wszystkie przeglądarki)	868
Składnia	869
O obiektach	869
Właściwości	870
Zastosowanie właściwości appVersion	870
Szczegółowe informacje o właściwości userAgent	874
Metody	886

Obiekt mimeType	889
Składnia	889
O obiekcie	889
Właściwości	890
Obiekt plugin	894
Składnia	894
O obiekcie	894
Właściwości	895
Metody	896
Wyszukiwanie typów MIME i plug-inów	897
Ogólne informacje o użyciu obiektów mimeType i plugin	898
Sprawdzanie dostępności typu MIME	899
Sprawdzanie dostępności plug-inu	899
Jednoczesne sprawdzanie plug-inu oraz typu MIME	900
Sterowanie ręczną instalacją plug-inów	901
Wykrywanie plug-inów w przeglądarkach Internet Explorer działających w systemach Windows	902
Obiekt screen	906
Składnia	907
O obiekcie	907
Właściwości	907
Obiekt userProfile	912
Składnia	912
O obiekcie	913
Metody	915
Rozdział 29. Obiekty zdarzeń	919
Dlaczego korzystamy ze zdarzeń?	920
Jakimi informacjami dysponują zdarzenia (oraz kiedy informacje te są dostępne)	921
Stacyczny obiekt Event	922
Propagacja zdarzeń	922
Propagacja zdarzeń w przeglądarce NN 4	923
Propagacja zdarzeń w przeglądarkach IE 4+	933
Propagacja zdarzeń w przeglądarkach NN 6+	938
Odwołania do obiektu event	943
Odwołania do obiektu event w przeglądarkach IE 4+	943
Odwołania do obiektu event w przeglądarkach NN 4+ (oraz DOM W3C)	943
Zgodność obiektu event	945
Konkurujące modele obiektu event	947
Sprawdzanie klawiszy modyfikatorów działające w różnych przeglądarkach	947
Pobieranie kodu klawisza w sposób działający w różnych przeglądarkach	948
Typy zdarzeń	950
Starsze wersje przeglądarek	950
Typy zdarzeń w przeglądarkach IE 4+ oraz NN 6	951
Obiekt event Netscape Navigatora 4	953
Składnia	953
O obiekcie	953
Właściwości	953
Obiekt event w przeglądarkach IE 4+	958
Składnia	959
O obiekcie	959
Właściwości	960

Obiekt event w przeglądarkach NN 6+	977
Składnia	978
O obiekcie	978
Właściwości	979
Metody	990
Rozdział 30. Obiekty stylów i arkuszy stylów.....	993
Wyjaśnienie nazw stosowanych obiektów	994
Importowane arkusze stylów	996
Odczytywanie właściwości stylów	996
Obiekt elementu STYLE	997
Składnia	997
O obiekcie	998
Właściwości	998
Obiekt styleSheet	999
Składnia	999
O obiekcie	1000
Właściwości	1000
Metody	1009
Obiekty cssRule oraz rule	1011
Składnia	1011
O obiektach	1011
Właściwości	1012
Obiekty currentStyle, runtimeStyle oraz style	1015
Składnia	1015
O obiektach	1015
Właściwości stylów	1016
Wartości właściwości	1018
Tekst i czcionki	1021
Wyświetlanie wewnątrzwierszowe i rozmieszczanie	1031
Właściwości związane z umiejscawianiem	1040
Właściwości tła	1043
Właściwości obramowań i krawędzi	1045
Właściwości list	1050
Właściwości pasków przewijania	1051
Właściwości tabel	1052
Właściwości strony i drukowania	1053
Pozostałe właściwości	1055
Właściwości związane z prezentacją dźwiękową	1056
Obiekt filter	1057
Składnia	1057
O obiekcie	1057
Zmiany sposobu zapisu filtrów w przeglądarce IE 5.5	1063
Rozdział 31. Obiekty umiejscowione	1071
Co to jest warstwa?	1072
Obiekt layer stosowany w przeglądarce NN 4	1073
Składnia	1073
O obiekcie	1074
Odwołania do warstw	1074
Warstwy i formularze	1075

Warstwy i tabele	1077
Właściwości	1077
Metody	1088
Procedury obsługi zdarzeń	1092
Elementy umiejscowione w nowoczesnym modelu obiektów dokumentu	1094
Zmiana tła elementu	1094
Przycinanie warstw	1097
Skryptowa obsługa warstw zagnieżdżonych	1102
Wyświetlanie w warstwie zawartości zewnętrznych dokumentów HTML	1109
Modyfikacja widoczności elementów umiejscawianych	1111
Modyfikacja kolejności wyświetlania warstw	1112
Przeciąganie i zmiana wielkości warstwy	1114
Rozdział 32. Osadzone obiekty — podsumowanie	1121
Obiekt dla elementu APPLET	1122
Składnia	1122
O obiekcie	1123
Właściwości	1123
Obiekt dla elementu OBJECT	1127
Składnia	1128
O obiekcie	1128
Właściwości	1129
Obiekt dla elementu EMBED	1134
Składnia	1134
O obiekcie	1135
Właściwości	1135
Dziwne zachowanie elementu PARAM	1137
Rozdział 33. Obiekty XML-a — podsumowanie	1139
Elementy i węzły	1140
Obiekt dla elementu XML	1142
Składnia	1142
O obiekcie	1142
Właściwości	1143
Część IV Opis języka JavaScript	1145
Rozdział 34. Obiekt String	1147
Typy łańcuchowe i typy liczbowe	1147
Proste łańcuchy znaków	1148
Tworzenie długich zmiennych łańcuchowych	1148
Łączenie literałów i zmiennych łańcuchowych	1149
Znaki specjalne wewnątrz łańcuchów znaków	1149
Obiekt String	1150
Składnia	1152
O obiekcie	1152
Właściwości	1153
Metody rozbioru łańcuchów	1156
Użyteczne funkcje operujące na łańcuchach znaków	1167
Metody formatujące	1168
Kodowanie i dekodowanie adresów URL	1170

Rozdział 35. Obiekty Math, Number i Boolean	1173
Liczby w języku JavaScript	1174
Liczby całkowite i zmiennopozycyjne	1174
Liczby całkowite w zapisie szesnastkowym i ósemkowym	1176
Przekształcanie ciągów znaków w liczby	1177
Przekształcanie liczb w ciągi znaków	1178
Kiedy liczba nie jest liczbą	1179
Obiekt Math	1180
Składnia	1180
O obiekcie	1180
Właściwości	1180
Metody	1181
Generowanie liczb losowych	1182
Skrót do obiektu Math	1183
Obiekt Number	1183
Składnia	1184
O obiekcie	1184
Właściwości	1184
Metody	1185
Obiekt Boolean	1187
Składnia	1188
O obiekcie	1188
Rozdział 36. Obiekt Date	1189
Strefy czasowe i czas GMT	1189
Obiekt Date	1191
Tworzenie obiektu reprezentującego datę	1191
Własne metody i właściwości obiektu	1193
Metody obiektu Date	1194
Dostosowywanie stref czasowych	1197
Daty jako ciągi znaków	1198
Formaty daty przyjazne dla starszych przeglądarek	1198
Więcej o przekształceniach	1199
Arytmetyka na datach i czasie	1200
Odliczanie dni	1202
Błędy i chochliki w obiekcie Date	1204
Zatwierdzanie poprawności dat w formularzach	1205
Rozdział 37. Obiekt Array	1209
Dane strukturalne	1209
Tworzenie pustej tablicy	1210
Wypełnianie tablicy	1212
Ułatwienia w tworzeniu tablic w języku JavaScript 1.2	1214
Usuwanie zawartości tablic	1214
Tablice równoległe	1215
Tablice wielowymiarowe	1218
Właściwości obiektu Array	1219
Metody obiektu Array	1221

Rozdział 38. Wyrażenia regularne i obiekt RegExp	1231
Wyrażenia regularne i wzorce	1231
Podstawy języka wyrażeń regularnych.....	1233
Proste wzorce	1233
Znaki specjalne.....	1234
Grupowanie i odwołania wsteczne.....	1237
Relacje pomiędzy obiektami	1237
Korzystanie z wyrażeń regularnych	1243
Czy odnaleziono pasujący fragment?.....	1243
Pobieranie informacji o odnalezionym fragmencie.....	1244
Zastępowanie łańcuchów	1246
Obiekt wyrażenia regularnego.....	1249
Składnia	1249
O obiekcie	1249
Właściwości	1250
Metody	1252
Obiekt RegExp	1254
Składnia	1254
O obiekcie	1254
Właściwości	1255
Rozdział 39. Struktury sterujące i obsługa wyjątków.....	1259
Podejmowanie decyzji przy użyciu instrukcji if oraz if...else	1260
Proste decyzje.....	1260
Kilka słów o wyrażeniach warunkowych.....	1261
Decyzje złożone	1262
Zagnieżdżanie instrukcji if...else.....	1263
Wyrażenia warunkowe	1265
Powtarzanie w pętłach for	1266
Wykorzystanie licznika pętli	1268
Przerywanie wykonywania pętli	1269
Sterowanie realizacją pętli przy użyciu instrukcji continue.....	1270
Pętla while	1271
Pętla do-while	1272
Pętle operujące na właściwościach (for-in).....	1273
Instrukcja with	1274
Instrukcje z etykietami.....	1275
Instrukcja switch.....	1278
Obsługa wyjątków	1280
Wyjątki i błędy	1280
Mechanizm obsługi wyjątków	1282
Wykorzystanie instrukcji try, catch i finally	1283
Praktyczne wykorzystanie obsługi wyjątków	1286
Zgłaszanie wyjątków	1287
Obiekt Error.....	1291
Składnia	1292
O obiekcie	1292
Właściwości	1292
Metody	1295

Rozdział 40. Operatory w języku JavaScript	1297
Kategorie operatorów	1297
Operatory porównania	1298
Porównywanie wartości różnych typów	1300
Operatory matrymonialne	1301
Operatory przypisania	1304
Operatory logiczne	1306
Operacje logiczne	1307
Przykłady praktycznego wykorzystania operatorów logicznych	1309
Operatory bitowe	1310
Operatory obiektowe	1311
Pozostałe operatory	1316
Priorytet operatorów	1319
Rozdział 41. Funkcje i obiekty niestandardowe	1323
Obiekt Function	1323
Składnia	1323
O obiekcie	1324
Tworzenie funkcji	1324
Zagnieżdżanie funkcji	1326
Argumenty funkcji	1326
Właściwości	1327
Metody	1331
Uwagi dotyczące wykorzystania funkcji	1333
Wywoływanie funkcji	1333
Zasięg zmiennych: zmienne globalne i lokalne	1334
Zmienne parametryczne	1338
Rekurencja w funkcjach	1338
Grupowanie funkcji w biblioteki	1339
Obiekty niestandardowe	1340
Przykład — obiekty planetarne	1341
Tworzenie tablic obiektów	1345
Dodawanie metod niestandardowych	1347
Inne sposoby tworzenia obiektów	1348
Metody śledzenia obiektów	1349
Definiowanie metod określających i odczytujących wartości właściwości	1350
Wykorzystanie obiektów niestandardowych	1353
Pojęcia obiektowe	1353
Dodawanie prototypu	1353
Dziedziczenie prototypowe	1354
Obiekty zagnieżdżone a dziedziczenie prototypowe	1354
Obiekt Object	1356
Składnia	1357
O obiekcie	1357
Metody	1357
Rozdział 42. Funkcje oraz instrukcje globalne	1359
Funkcje	1360
Instrukcje	1369
Obiekty charakterystyczne dla IE/Windows	1372

Dodatki.....	1379
Dodatek A Opis obiektów języka JavaScript oraz obiektów przeglądarki ...	1381
Dodatek B Zarezerwowane identyfikatory JavaScriptu	1395
Dodatek C Odpowiedzi do ćwiczeń z samouczka.....	1397
Odpowiedzi do rozdziału 4.....	1397
Odpowiedzi do rozdziału 5.....	1398
Odpowiedzi do rozdziału 6.....	1399
Odpowiedzi do rozdziału 7.....	1400
Odpowiedzi do rozdziału 8.....	1404
Odpowiedzi do rozdziału 9.....	1405
Odpowiedzi do rozdziału 10.....	1408
Odpowiedzi do rozdziału 11.....	1410
Odpowiedzi do rozdziału 12.....	1410
Dodatek D Zasoby w Internecie dotyczące JavaScriptu i DOM	1411
Pomoce i uaktualnienia do tej książki	1411
Grupy dyskusyjne.....	1411
FAQ-i.....	1412
Dokumentacja online.....	1413
Sieć WWW	1413
Skorowidz	1415

Rozdział 38.

Wyrażenia regularne i obiekt RegExp

W tym rozdziale:

- ◆ Czym są wyrażenia regularne?
- ◆ Jak używać wyrażeń regularnych w operacjach typu wyszukaj i zastąp?
- ◆ Jak używać wyrażeń regularnych wraz z metodami obiektu String?

Programiści którzy używali języka Perl (lub innych języków programowania wykorzystywanych do tworzenia aplikacji internetowych) wiedzą, że wyrażenia regularne niezwykle ułatwiają przetwarzanie odbieranych danych oraz ich formatowania przed wyświetleniem na stronach WWW lub zapisaniem w bazach danych na serwerze. Dzięki niezwyklej elastyczności i spójności wyrażeń regularnych wszelkie przekształcenia wymagające wielokrotnego przeszukiwania i zastępowania tekstu można znacznie uprościć i udoskonalić. W interpreterze JavaScriptu wyrażenia regularne dostępne są począwszy od Navigатора 4 i Internet Explorera 4 (choć w Internet Explorerze 5 możliwości wykorzystania wyrażeń regularnych zostały znacznie rozbudowane).

Wyrażenia regularne najbardziej przydadzą się osobom tworzącym programy CGI działające na serwerach WWW wyposażonych w interpreter JavaScriptu, w wersji, która umożliwia wykorzystanie wyrażeń regularnych. Nie wyklucza to jednak wykorzystania tego swoistego „języka w języku” przy tworzeniu aplikacji działających po stronie klienta. Jeśli tworzone skrypty przeprowadzają weryfikację danych lub jakiegokolwiek inne zaawansowane operacje związane z przetwarzaniem i analizą tekstów, to należy zastanowić się nad wykorzystaniem wyrażeń regularnych zamiast stosować stosunkowo złożone funkcje języka JavaScript.

Wyrażenia regularne i wzorce

W kilku wcześniejszych rozdziałach niniejszej książki opisywałem wyrażenia jako sekwencję identyfikatorów, słów kluczowych i (lub) operatorów, które po przetworzeniu przez interpreter JavaScriptu zwracają jakąś wartość. Wyrażenie regularne funkcjonuje zgodnie z tym opisem, daje jednak znacznie większe możliwości. Najważniejsze jest to,

że wyrażenie regularne wykorzystuje sekwencję znaków i symboli, za pomocą których definiowany jest wzorzec tekstu. Wzorzec ten wykorzystywany jest do zlokalizowania fragmentu tekstu spełniającego zadane kryteria.

Doświadczeni programiści mogliby w tym miejscu zauważyć, że język JavaScript udostępnia przecież metody `łańcuch.indexOf()` oraz `łańcuch.lastIndexOf()`, które bardzo szybko mogą określić czy podany łańcuch znaków jest częścią innego łańcucha, a nawet w jakim jego miejscu się znajduje. Metody te można z powodzeniem zastosować w sytuacjach, gdy poszukujemy ściśle określonego łańcucha znaków. Jeśli jednak należy przeprowadzić bardziej wyszukaną operację (na przykład, sprawdzić czy łańcuch znaków zawiera pięciocyfrowy kod pocztowy), to wygodne metody obiektu `String` na nic się nam nie przydadzą — musielibyśmy napisać własne funkcje umożliwiające analizę łańcucha znaków. I tu jednak uwidacznia się całe piękno wyrażen regularnych: pozwalają one na zdefiniowanie łańcucha znaków, który będzie używany przy wyszukiwaniu i na podstawie podanych wskazówek jest w stanie samodzielnie i „inteligentnie” określić co odpowiada zadanym kryteriom.

Najprostsze z możliwych wyrażen regularnych niczym się nie różnią od łańcuchów znaków używanych w metodzie `łańcuch.indexOf()`. Taki wzorzec jest po prostu łańcuchem znaków, który chcemy odszukać. W języku JavaScript jednym ze sposobów tworzenia wyrażen regularnych jest zapisane wyrażenia pomiędzy dwoma znakami ukośnika. Na przykład, założmy, że dysponujemy następującym łańcuchem znaków:

```
0. witam, czy masz ochotę zagrać Otella w szkolnym przedstawieniu?
```

Zarówno powyższy, jak i inne łańcuchy znaków mogą być przeszukiwane przez skrypt, którego zadaniem jest zamiana formalnych zwrotów grzecznościowych na określenia bardziej kolokwialne. A zatem, jednym z zadań tego skryptu jest zastępowanie słowa „witam” słowem „cześć”. Typowy algorytm wyszukiwania i zamiany, określanym jako „algorytm brutalnej siły”, rozpoczyna się od zdefiniowania prostego wzorca, który będzie używany przy wyszukiwaniu. W języku JavaScript taki wzorzec (wyrażenie regularne) definiuje się poprzez zapisanie łańcucha znaków pomiędzy dwoma znakami ukośnika. Osobiście, ze względu na wygodę oraz w celu zachowania przejrzystości skryptu, zazwyczaj zapisuję wyrażenia regularne w zmiennych, tak jak to pokazałem na poniższym przykładzie:

```
var myRegularExpression = /witam/
```

W połączeniu z odpowiednimi metodami wyrażen regularnych lub obiektu `String`, powyższy wzorzec umożliwi odszukanie łańcucha znaków „witam” w dowolnym miejscu przeszukiwanego łańcucha. Problem polega na tym, że ten prosty wzorzec przysparza poważnych problemów podczas operacji wyszukiwania i zastępowania wykonywanej w pętli. Wzorzec ten odnajduje bowiem nie tylko słowo „witam”, lecz także „zakwitam”, „powitam” i inne.

Próba napisania kolejnej procedury, która metodą brutalnej siły próbowałaby odnaleźć i zastąpić wyłącznie całe słowa, a nie ich fragmenty, mogłaby się okazać prawdziwym koszmarem. Nie wystarczy bowiem zmodyfikować wzorca przez dodanie przed nim lub za nim (bądź też po obu jego stronach) znaku odstępu, gdyż zarówno przed, jak i za wyrazem może się pojawić dowolny znak przestankowy, na przykład: kropka, przecinek bądź średnik. Na szczęście wyrażenia regularne udostępniają skrócony sposób pozwalający na

określenie ogólnych cech, w tym także tak zwanej „granicy wyrazu”. Symbolem określającym „granice wyrazu” jest `\b` (odwrotny ukośnik oraz mała litera b). Jeśli zmienimy definicję wyrażenia regularnego, tak aby po obu stronach poszukiwanego wyrazu znalazł się symbol granicy wyrazu, to instrukcja tworząca takie wyrażenie regularne będzie mieć następującą postać:

```
var myRegularExpression = /\bwitam\b/
```

Gdy interpreter JavaScriptu użyje tego wyrażenia jako argumentu wywołania metody obiektu `String` realizującej operację wyszukiwania i zastępowania, to w rezultacie zostaną zmienione wyłącznie całe słowa „witam”; inne słowa, w których występuje łańcuch znaków `witam` zostaną pominięte.

Jeśli dopiero uczysz się języka JavaScript i nie dysponujesz żadnymi doświadczeniami związanymi z wykorzystaniem wyrażeń regularnych w innych językach programowania, to za wykorzystanie ich ogromnych możliwości będziesz niestety musiał sporo zapłacić — trzeba się powiem nauczyć języka tworzenia wyrażeń regularnych, który używa tylu symboli, że wyrażenia czasami wyglądają jak zbitki bezsensownych znaków, używane w komiksach zamiast przekleństw. Niniejszy rozdział został pomyślany jedynie jako przedstawienie składni używanej w języku JavaScript do tworzenia wyrażeń regularnych, a nie jako wyczerpujący podręcznik. Duże znaczenie ma zrozumienie sposobu traktowania obiektów, którymi są wyrażenia regularne przez interpreter JavaScriptu oraz różnic pomiędzy tymi obiektami a statycznym obiektem `RegExp`. Mam nadzieję, że przykłady zamieszczone w niniejszym rozdziale pokażą ci choć część ogromnych możliwości, jakie dają wyrażenia regularne. Wyczerpujące omówienie możliwości oraz tajników tworzenia wyrażeń regularnych można znaleźć w książce „Wyrażenia regularne” autorstwa Jeffreya E.F. Fridla, wydanej przez wydawnictwo Helion.

Podstawy języka wyrażeń regularnych

Aby wyczerpująco przedstawić składnię wyrażeń regularnych, całość zagadnienia podzieliłem na trzy części. Pierwsza z nich poświęcona jest prostym wyrażeniom (których przykłady już widziałeś). Następnie zaprezentuję więcej znaków specjalnych używanych podczas definiowania specyfikacji łańcucha wyszukiwanego. Ostatnim zagadnieniem będzie użycie nawiasów, które nie tylko ułatwiają grupowanie wyrażeń w celu zmiany kolejności ich wykonywania, lecz także pozwalają na tymczasowe przechowywanie pośrednich wyników złożonych wyrażeń i wykorzystanie ich do modyfikacji łańcucha po jego podzieleniu według kryteriów określonych przez wyrażenie regularne.

Proste wzorce

Proste wyrażenia regularne, to wyrażenia, w których do definiowania łańcucha używanego w wyszukiwaniu, nie są używane żadne znaki specjalne. A zatem, aby zamienić każdy znak odstępu na znak podkreślenia, można użyć prostego wzorca przedstawionego poniżej, który odnajduje znaki odstępu:

```
var re = / /
```

W powyższym wyrażeniu, pomiędzy znakami ukośnika wyznaczającymi początek i koniec wyrażenia regularnego, został umieszczony znak odstępu. Jednak wyrażenie to nie całkiem spełnia nasze oczekiwania, gdyż za jego pomocą można odszukać wyłącznie jeden znak odstępu w całym, długim łańcuchu znaków. Jednak można nakazać, aby wyrażenie regularne operowało globalnie — za pomocą podanego wzorca do przeszukania całego łańcucha znaków. W tym celu, do wyrażenia regularnego należy dodać modyfikator `g`:

```
var re = / /g
```

Gdy wartość `re` zostanie przekazana jako argument wywołania metody `replace()` wykorzystującej wyrażenie regularne (opisanej w dalszej części rozdziału), zastąpione zostaną wszystkie odnalezione fragmenty przeszukiwanego łańcucha, a nie tylko pierwszy z nich. Należy zwrócić uwagę, że modyfikator umieszczany jest za drugim znakiem ukośnika, kończącym wyrażenie regularne.

W wyrażeniach regularnych — podobnie jak w wielu innych operacjach na łańcuchach znaków wykonywanych w języku JavaScript — uwzględniana jest wielkość liter. Zachowanie to można zmienić przez zastosowanie odpowiedniego modyfikatora. A zatem, poniższe wyrażenie:

```
var re = /web/i
```

odnajdzie zarówno słowo „web”, jak i „Web”, oraz słowo to zapisane za pomocą dowolnej innej kombinacji dużych i małych liter. Oba modyfikatory można połączyć i zapisać je na końcu wyrażenia regularnego. Na przykład, poniższe wyrażenie wyszukuje słowo „web” bez względu na wielkość liter i operuje na całym przeszukiwanym łańcuchu znaków:

```
var re = /web/gi
```

Zgodnie z założeniami trzeciej wersji standardu ECMA-262, zarówno przeglądarka Internet Explorer 5.5, jak i Netscape Navigator 6 udostępniają także modyfikator wymuszający przeszukiwanie wielu wierszy tekstu w długich łańcuchach znaków (czyli łańcuchach zawierających znaki powrotu karetki). Modyfikator ten jest oznaczany jako `m`.

Znaki specjalne

Większość składowych wyrażeń regularnych w języku JavaScript bazuje na wyrażeniach regularnych języka Perl. W kilku przypadkach JavaScript udostępnia alternatywne rozwiązania umożliwiające uproszczenie składni wyrażeń lub posłużyć się składnią stosowaną w języku Perl, co niewątpliwie docenią osoby znające ten język.

Użycie wyrażeń regularnych daje programistom duże pole manewru, pozwalając w zwięzły sposób określić takie cechy wyszukiwanego łańcucha znaków jak: typy znaków, jakie mogą się w nim pojawić, znaki, jakie mogą się pojawić na jego krańcach oraz jak często łańcuch ten może się pojawiać. Grupa specjalnych, jednoznakowych symboli sterujących (czyli liter poprzedzanych znakiem odwrotnego ukośnika) obsługuje wszelkie możliwości związane z wyszukiwaniem znaków, z kolei symbole przestankowe oraz grupujące umożliwiają zdefiniowanie zagadnień związanych z częstością wyszukiwania i zasięgiem.

We wcześniejszej części rozdziału przedstawiłem przykład pokazujący symbol `\b`, który określał granice wyrazu na jednym z krańców łańcucha wyszukiwanego. Wszystkie symbole specjalne, które można stosować w wyrażeniach regularnych w języku JavaScript zostały przedstawione w tabeli 38.1. Podane symbole zaliczane są do metaznaków — czyli tych części wyrażeń, które nie stanowią części poszukiwanego łańcucha, lecz raczej pełnią funkcję poleceń lub wskazówek odnośnie do działania wyrażenia regularnego.

Tabela 38.1. Metaznaki używane przy tworzeniu wyrażeń regularnych w języku JavaScript

Znak	Co wskazuje	Przykład
<code>\b</code>	Granice słowa	<code>/\bor/</code> wskazuje na przykład wyrazy <code>origami</code> lub <code>or</code> , lecz nie <code>normalny</code>
<code>\B</code>	Wszystko z wyjątkiem granicy słowa	<code>/or\b/</code> wskazuje na przykład wyrazy <code>motor</code> lub <code>or</code> , lecz nie <code>platforma</code>
<code>\d</code>	Cyfrę od 0 do 9	<code>/\d\d\d/</code> wskazuje na przykład ciąg <code>234</code> lub <code>493</code> , lecz nie <code>B12</code>
<code>\D</code>	Wszystko z wyjątkiem cyfry	<code>/\D\D\D/</code> wskazuje na przykład ciąg <code>ABC</code> , lecz nie <code>212</code> lub <code>B17</code>
<code>\s</code>	Pojedynczy znak odstępu	<code>/over\sbite/</code> wskazuje na przykład ciąg <code>over bite</code> , lecz nie <code>overbite</code> ani <code>over bite</code>
<code>\S</code>	Pojedynczy znak rozdzielający, ale nie odstęp	<code>/over\Sbite/</code> wskazuje na przykład ciąg <code>over-bite</code> , lecz nie <code>overbite</code> lub <code>over bite</code>
<code>\w</code>	Literę, cyfrę lub znak podkreślenia	<code>/A\w/</code> wskazuje na przykład ciąg <code>A1</code> lub <code>AA</code> , lecz nie <code>A+</code>
<code>\W</code>	Wszystko z wyjątkiem litery, cyfry lub znaku podkreślenia	<code>/A\W/</code> wskazuje na przykład ciąg <code>A+</code> , lecz nie ciągi <code>AA</code> i <code>A1</code>
<code>.</code>	Dowolny znak z wyjątkiem znaku nowego wiersza	<code>/.../</code> wskazuje ciągi <code>ABC</code> , <code>1+3</code> , <code>A 4</code> oraz wszelkie inne możliwe ciągi składające się z trzech znaków
<code>[...]</code>	Zbiór znaków	<code>/[AN]BC/</code> wskazuje ciągi <code>ABC</code> i <code>NBC</code> , lecz nie <code>BBC</code>
<code>[^...]</code>	Wszystko z wyjątkiem podanego zbioru znaków	<code>/[^AN]BC/</code> wskazuje ciągi <code>BBC</code> lub <code>CBC</code> , lecz nie <code>ABC</code> lub <code>NBC</code>

Metaznaków przedstawionych w tabeli 38.1 nie należy mylić ze znakami sterującymi oznaczającymi znaki: tabulacji (`\t`), nowego wiersza (`\n`), powrotu karetki (`\r`), przewinięcia wiersza (`\f`) oraz tabulacji pionowej (`\v`).

Przyjrzyjmy się teraz dokładniej metaznakom `[...]` oraz `[^...]`. Wewnątrz tych nawiasów kwadratowych można podać pojedyncze znaki (jak to pokazałem na przykładach w tabeli 38.1), ciągły zakres znaków lub kombinację obu tych rozwiązań. Na przykład, metaznak `\d` można także zdefiniować jako `[0-9]`, co oznacza dowolny znak z zakresu od 0 do 9. Aby wskazać cyfrę 2 lub dowolną cyfrę z zakresu od 6 do 8, należałoby użyć zapisu `[26-8]`. Podobnie, metaznak `\w` można wyrazić jako `[A-Za-z0-9_]`, przy czym należy pamiętać o tym, że w wyrażeniach regularnych jest uwzględniana wielkość liter.

Wszystkie wyrażenia przedstawione w tabeli 38.1, z wyjątkiem wyrażeń zapisanych w nawiasach kwadratowych, wskazują na pojedynczy znak. W większości wypadków nie można jednak przewidzieć w jaki sposób sformatowane będą dane przychodzące —

nie znamy bowiem ani długości słowa, ani liczby cyfr w liczbie. Kilka kolejnych metaznaków umożliwia określenie częstotliwości wystąpienia pojedynczego znaku lub całego typu znaków (określonego podobnie jak w tabeli 38.1). Jeśli dysponujesz jakimiś doświadczeniami w obsłudze systemów operacyjnych wykorzystujących wiersz poleceń, przekonasz się, że występują pewne analogie pomiędzy znakami wieloznacznymi i metaznakami stosowanymi w wyrażeniach regularnych. Metaznaki umożliwiające określanie częstotliwości występowania znaków lub fragmentów wyrażeń regularnych języka JavaScript zostały przedstawione w tabeli 38.2.

Tabela 38.2. Metaznaki określające liczbę wystąpień w wyrażeniach regularnych JavaScriptu

Znak	Wskazuje ostatni raz	Przykład
*	Zero lub więcej razy	<code>/Ja*vaScript/</code> wskazuje łańcuchy <code>JvaScript</code> , <code>JavaScript</code> lub <code>JaaavaScript</code> , lecz nie <code>JvaScript</code>
?	Zero lub dokładnie raz	<code>/Ja?vaScript/</code> wskazuje łańcuchy <code>JvaScript</code> lub <code>JavaScript</code> , lecz nie <code>JaaavaScript</code>
+	Jeden lub więcej razy	<code>/Ja+vaScript/</code> wskazuje łańcuch <code>JavaScript</code> lub <code>JaaavaScript</code> , lecz nie <code>JvaScript</code>
{ <i>n</i> }	Dokładnie <i>n</i> razy	<code>/Ja{2}vaScript/</code> wskazuje łańcuch <code>JaavaScript</code> , lecz nie <code>JvaScript</code> lub <code>JavaScript</code>
{ <i>n</i> ,}	<i>N</i> lub więcej razy	<code>/Ja{2,}vaScript/</code> wskazuje łańcuchy <code>JaavaScript</code> lub <code>JaaavaScript</code> , lecz nie <code>JavaScript</code>
{ <i>n</i> , <i>m</i> }	Co najmniej <i>n</i> razy, lecz co najwyżej <i>m</i> razy	<code>/Ja{2,3}vaScript/</code> wskazuje łańcuchy <code>JaavaScript</code> oraz <code>JaaavaScript</code> , lecz nie <code>JavaScript</code>

Każdy z metaznaków przedstawionych w tabeli 38.2 odnosi się do znaku podanego w wyrażeniu regularnym bezpośrednio przed danym metaznakiem. Warto zauważyć, że znakami tymi mogą także być metaznaki podane w tabeli 38.1. Na przykład, poniższe wyrażenie wskazuje ciągi zawierające dwie cyfry oddzielone od siebie jedną lub kilkoma samogłoskami:

```
\/d[aeiou]+d\/
```

Ostatnią, niezwykle ważną możliwością, jaką dają nam metaznaki stosowane w wyrażeniach regularnych, jest określanie położenia, w jakim musi występować poszukiwany łańcuch znaków. W tym przypadku określenia „położenie” nie należy utożsamiać z przesunięciem od początku łańcucha — te informacje zwraca bowiem sam mechanizm obsługi wyrażeń regularnych. W tym przypadku chodzi o określenie czy poszukiwany łańcuch znaków ma się znajdować na początku lub na końcu wiersza (jeśli ma to jakieś znaczenie) lub innego łańcucha wskazanego jako główny łańcuch do przeszukania.

Tabela 38.3. Metaznaki określające położenie w wyrażeniach regularnych JavaScriptu

Znak	Lokalizacja szukanego łańcucha	Przykład
^	Na początku przeszukiwanego łańcucha lub wiersza	Wyrażenie <code>/^Fred/</code> wskazuje <code>Fred jest super</code> , lecz nie <code>Czy Fred jest tutaj?</code>
\$	Na końcu przeszukiwanego łańcucha lub wiersza	Wyrażenie <code>/Fred\$/</code> wskazuje <code>To jest Fred</code> , lecz nie <code>Fred jest super</code> lub <code>Czy Fred jest tutaj?</code>

Na przykład, celowe może być wskazywanie cyfr rzymskich tylko w przypadkach, gdy zostały one umieszczone na samym początku wiersza, a nie gdzieś w jego dalszej części. Jeśli cyfry rzymskie są w dokumencie używane do numerowania listy, to wszystkie jej punkty można wskazać za pomocą poniższego wyrażenia:

```
/^[IVXMDCL]+\./
```

Powyższe wyrażenie regularne wskazuje dowolną kombinację cyfr rzymskich zakończonych kropką (w wyrażeniach regularnych kropka jest znakiem specjalnym, zgodnie z informacjami podanymi w tabeli 38.1, a zatem, aby użyć kropki jako jednego z wyszukiwanych znaków, należy ją poprzedzić znakiem odwrotnego ukośnika). Dodatkowo, cyfry rzymskie muszą być umieszczone na samym początku wiersza — przed nimi nie mogą się znajdować nawet żadne odstępów ani znaki tabulacji. A zatem, łańcuch znaków Patrz część VI nie zostanie wskazany przez powyższe wyrażenie, gdyż liczba rzymska nie znajduje się na samym początku wiersza.

Skoro już mówimy o wierszach, to określimy co jest rozumiane pod tym pojęciem. Otóż wiersz to ciągle łańcuch znaków zakończony znakiem nowego wiersza i (lub) powrotu karetki (w zależności od używanego systemu operacyjnego). Jeśli w wielowierszowym polu tekstowym zostanie włączona opcja przenoszenia wyrazów do nowego wiersza, nie będzie ona miała wpływu na faktyczne miejsca, gdzie kończą się poszczególne wiersze tekstu.

Grupowanie i odwołania wsteczne

Wyrażenia regularne zachowują zgodność z większością obowiązujących w JavaScriptcie reguł dotyczących zmiany kolejności wyliczania wyrażeń w przypadku grupowania ich przy użyciu nawiasów oraz logicznego operatora Or. Jedną z różnic polega na tym, że w wyrażeniach regularnych operator logiczny Or zapisywany jest jako pojedyncza pionowa kreska (`|`), a nie jak w języku JavaScript jako dwie pionowe kreski.

Nawiasy stosowane w wyrażeniach regularnych nie tylko służą do grupowania fragmentów wyrażeń i zmiany kolejności ich wyliczania. Każdy nawias umożliwia zapamiętanie fragmentu przeszukiwanego łańcucha pasującego do fragmentu wyrażenia regularnego zapisanego w danym nawiasie. Zapisywanie wskazanego fragmentu łańcucha jest realizowane automatycznie, przy czym poszczególne łańcuchy są przechowywane w tablicy indeksowanej liczbami, do której dostęp ma zarówno samo wyrażenie regularne, jak i skrypt (choć w obu tych przypadkach odwołania do tablicy tworzone są w inny sposób). Korzystanie z tych zapamiętanych fragmentów określa się mianem odwołań wstecznych, gdyż wyrażenie regularne może się dzięki nim odwołać do wyników zwróconych przez fragment zdefiniowany we wcześniejszej części wyrażenia. Te zapamiętane fragmenty wyrażeń bardzo się przydają podczas wykonywania operacji zamiany, które zostaną omówione w dalszej części rozdziału.

Relacje pomiędzy obiektami

W języku JavaScript podczas tworzenia nawet najprostszych wyrażeń regularnych i wykonywania operacji przy ich wykorzystaniu, bardzo wiele czynności jest realizowanych w sposób niewidoczny dla programisty. Choć język używany do tworzenia wyrażeń regularnych

przedstawiony we wcześniejszej części rozdziału jest bardzo ważny, aby w pełni wykorzystać możliwości, jakie dają wyrażenia regularne, ważniejszym zagadnieniem jest poznanie wzajemnych relacji pomiędzy obiektami JavaScriptu.

W pierwszej kolejności należy sobie uświadomić, że mamy do czynienia z dwoma odrębnymi elementami — obiektem wyrażenia regularnego oraz statycznym obiektem `RegExp`. Oba te obiekty należą do jądra języka JavaScript i nie wchodzą w skład modelu obiektów dokumentu. Obiekty te współpracują ze sobą, lecz mają całkowicie odrębne zbiory właściwości.

Tworząc wyrażenie regularne (nawet w przypadku, gdy jest w tym celu wykorzystywany zapis `/.../`), JavaScript wywołuje konstruktor `new RegExp()`, analogicznie do sposobu, w jaki konstruktor `new Date()` tworzy obiekt zawierający informacje o konkretnej dacie. Obiekt wyrażenia regularnego zwrócony przez konstruktor zawiera kilka właściwości przechowujących szczegółowe informacje na temat zapamiętanych w wyrażeniu danych. Z kolei obiekt `RegExp` jest statyczny i istnieje tylko jedna kopia tego obiektu, która zawiera odrębne właściwości monitorujące działania wyrażenia regularnego w bieżącym oknie przeglądarki (lub ramce).

Aby pokazać operacje wykonywane zazwyczaj w sposób całkowicie niewidoczny, krok po kroku przedstawię proces tworzenia i wykorzystania wyrażeń regularnych. Pokażę co dzieje się z właściwościami powiązanych ze sobą obiektów w sytuacji, gdy spróbujemy odszukać łańcuch znaków w wyniku wywołania w tym celu jednej z metod wyrażenia regularnego.



Kilka właściwości obiektu wyrażenia regularnego oraz statycznego obiektu `RegExp` prezentowanych w dalszej części rozdziału, jest dostępnych w przeglądarce Internet Explorer w wersji 5.5 lub późniejszej. Wszystkie te właściwości są natomiast dostępne w przeglądarkach NN 4+. Informacje o dostępności tych właściwości w konkretnych wersjach przeglądarek zostały podane w dalszej części rozdziału, podczas prezentacji poszczególnych właściwości.

Punktem wyjścia będzie fragment monologu Hamleta (w wersji oryginalnej), zapisany w zmiennej o nazwie `mainString`:

```
var mainString = "To be, or not to be: That is the question:"
```

Przy założeniu, że naszym ostatecznym celem jest wyszukanie wszystkich łańcuchów znaków `be`, w pierwszej kolejności należy stworzyć wyrażenie regularne odpowiadające temu słowu. Definiując wyrażenie, określiłem, że ma ono być wyszukiwane globalnie (sam obiekt wyrażenia regularnego jest zapisywany w zmiennej `re`):

```
var re = /\bbe\b/
```

Aby zagwarantować, że wyrażenie wskaże wyłącznie samodzielne wyrazy `be`, definiując wyrażenia, zapisałem te litery pomiędzy metaznakami określającymi granicę słowa. Na końcu wyrażenia dodałem modyfikator `g`. Zmienna, do której zostało przypisane wyrażenie — `re` — reprezentuje obiekt wyrażenia regularnego, posiadający następujące właściwości i wartości:

Ostatnia grupa właściwości (o nazwach od \$1 do \$9) służy do przechowywania odwołań wstecznych. Jednak w zdefiniowanym wcześniej wyrażeniu regularnym nie ma żadnych nawiasów, a zatem właściwości te są puste i pominałem je w kolejnych prezentowanych zestawieniach.

Dysponując gotowym do użycia obiektem wyrażenia regularnego, wywołuję jego metodę `exec()`, która przegląda główny łańcuch znaków w poszukiwaniu fragmentu pasującego do wzorca zdefiniowanego w wyrażeniu. Jeśli metoda odzyska wzorzec, zwraca kolejny obiekt, którego właściwości zawierają wiele informacji na temat odzyskanego łańcucha (w naszym przykładzie obiekt ten zapisuję w zmiennej `foundArray`):

```
var foundArray = re.exec(mainString)
```

JavaScript pozwala skrócić wywołanie metody `exec()`, jeśli obiekt wyrażenia regularnego potraktujemy jako metodę:

```
var foundArray = re(mainString)
```

Zazwyczaj przed wykonaniem dalszych czynności i sprawdzeniem właściwości powiązanych obiektów skrypt powinien sprawdzić, czy metoda `exec()` nie zwróciła wartości `null` (co by oznaczało, że podany wzorzec nie został odnaleziony). Ponieważ my przeprowadzamy eksperyment kontrolowany, doskonale wiemy, że w łańcuchu głównym istnieje przynajmniej jeden łańcuch pasujący do podanego wzorca, dlatego też w pierwszej kolejności przedstawię inne uzyskane wyniki. Wywołanie tej prostej metody nie tylko zwróciło dane zapisane w zmiennej `foundArray`, lecz także doprowadziło do modyfikacji wartości obiektu `RegExp` oraz obiektu wyrażenia regularnego. Bieżący stan obiektu wyrażenia regularnego przedstawiłem w poniższej tabelce:

Obiekt.nazwaWłaściwości	Wartość
<code>re.source</code>	"\bbe\b"
<code>re.global</code>	true
<code>re.ignoreCase</code>	false
<code>re.lastIndex</code>	5

W powyższym obiekcie tylko jedna zmiana odgrywa istotną rolę — wartość właściwości `lastIndex` podskoczyła do 5. Innymi słowy, wywołanie metody `exec()` musiało odnaleźć fragment łańcucha głównego pasujący do podanego wzorca, którego przesunięcie względem początku łańcucha powiększone o długość powoduje, że kolejne wyszukiwanie rozpocznie się od znaku łańcucha głównego o indeksie 5. Dokładnie w tym właśnie miejscu, w łańcuchu głównym znajduje się przecinek (po pierwszym słowie `be`). Gdyby w wyrażeniu regularnym nie został użyty modyfikator `g`, to właściwość `lastIndex` wciąż miałaby wartość 0, gdyż interpreter założyłby, że nie mają być wykonywane żadne kolejne wyszukiwania.

W wyniku wykonania metody `exec()` w wielu właściwościach obiektu `RegExp` zostały zapisane informacje dotyczące wyników wyszukiwania:

Obiekt.nazwaWłaściwości	Wartość
RegExp.input	
RegExp.multiline	false
RegExp.lastMatch	"be"
RegExp.lastParen	
RegExp.leftContext	"To "
RegExp.rightContext	", or not to be: That is the question:"

Z tego obiektu można odczytać odnaleziony łańcuch znaków pasujący do definicji wyrażenia regularnego. Dostępne są także segmenty głównego łańcucha znaków znajdujące się przed oraz za odnalezionym ciągiem (w naszym przykładzie, we właściwości `leftContext` za słowem `To` znajduje się jeszcze znak odstępu). Przeglądając tablicę zwróconą przez wywołanie metody `exec()` możemy się przekonać, iż mamy możliwość uzyskania dostępu do kolejnych informacji:

Obiekt.nazwaWłaściwości	Wartość
<code>foundArray[0]</code>	"be"
<code>foundArray.index</code>	3
<code>foundArray.input</code>	"To be, or not to be: That is the question:"

Pierwszym elementem tablicy posiadającym indeks 0, jest odnaleziony łańcuch znaków pasujący do wyrażenia regularnego. A zatem, element ten ma tę samą wartość, co właściwość `RegExp.lastMatch`. Dostępna jest także pełna wersja łańcucha głównego, przechowywana we właściwości `input`. Informacją, która może mieć duże znaczenie dla skryptu jest indeks określający w jakim miejscu łańcucha głównego odnaleziono fragment pasujący do wyrażenia regularnego. Informacja ta jest zapisywana we właściwości `index`. Dysponując tym indeksem oraz pozostałymi informacjami przechowywanymi w tablicy danych odnalezionych, możemy uzyskać te same łańcuchy, które są przechowywane we właściwościach `RegExp.leftContext` (ten łańcuch można pobrać przy użyciu wyrażenia `foundArray.input.substring(0, foundArray.index)`) oraz `RegExp.rightContext` (z kolei ten łańcuch można pobrać przy użyciu wyrażenia `foundArray.input.substring(foundArray.index, foundArray[0].length)`).

Ponieważ definiując wyrażenie regularne zażądałem, aby miało ono zasięg globalny (dodając do niego modyfikator `g`), więc, bez konieczności wprowadzania jakichkolwiek zmian, mogę ponownie wywołać metodę `exec()`. Choć kolejne wywołanie tej metody nie różni się od pierwszego wywołania, metoda rozpoczyna przeszukiwanie łańcucha głównego od miejsca wskazanego przez nową wartość właściwości `re.lastIndex`. Na skutek tego kolejnego wywołania metody `exec()` zmianie ulegną właściwości wszystkich trzech powiązanych ze sobą obiektów:

```
var foundArray = re.exec(mainString)
```

Wyniki drugiego wywołania metody `exec()` przedstawiłem w poniższej tabeli:

Obiekt.nazwaWłaściwości	Wartość
re.source	"\bbe\bq"
re.global	true
re.ignoreCase	false
re.lastIndex	19
RegExp.input	
RegExp.multiline	false
RegExp.lastMatch	"be"
RegExp.lastParen	
RegExp.leftContext	", or not to "
RegExp.rightContext	": That is the question:"
foundArray[0]	"be"
foundArray.index	17
foundArray.input	"To be, or not to be: That is the question:"

Z uwagi na odnalezienie kolejnego fragmentu pasującego do wyrażenia regularnego, metoda `exec()` ponownie zwraca tablicę `foundArray` zawierającą nowe dane. Właściwość `index` tej tablicy wskazuje teraz miejsce łańcucha głównego, w którym rozpoczyna się odnaleziony fragment pasujący do wyrażenia regularnego. Z kolei właściwość `lastIndex` obiektu wyrażenia regularnego wskazuje w jakim miejscu rozpocznie się kolejne wyszukiwanie (bezpośrednio za drugim słowem `be`). Odpowiednio zostały także zmodyfikowane wartości właściwości `RegExp.leftContext` oraz `RegExp.rightContext`.

Gdyby wzorzec zdefiniowany za pomocą wyrażenia regularnego nie określał jednego, wybranego słowa, to także niektóre inne właściwości tych trzech obiektów mogłyby ulec zmianie. Na przykład, gdyby wyrażenie regularne zawierało definicję formatu stosowanego do zapisywania kodu pocztowego, to właściwości `RegExp.lastMatch` oraz `foundArray[0]` zawierałyby aktualną odnalezioną wartość kodu, która zapewne za każdym razem byłaby inna.

Po kolejnym — trzecim — wywołaniu metody `exec()` w głównym łańcuchu znaków (`mainString`) nie zostanie już odnaleziony kolejny fragment pasujący do wyrażenia regularnego. Z tego faktu nie wynikają jednak żadne komplikacje. Przede wszystkim, tym razem metoda `exec()` zwróci wartość `null`, sygnalizując w ten sposób skryptowi, że nie został już odnaleziony żaden fragment pasujący do wyrażenia regularnego. Właściwość `lastIndex` obiektu wyrażenia regularnego ponownie przyjmuje wartość `0`, przygotowując w ten sposób rozpoczęcie przeszukiwania od początku innego łańcucha znaków. Jednak najważniejsze jest to, iż właściwości obiektu `RegExp` zachowują swoje wcześniejsze wartości. Dzięki temu, jeśli umieścimy wywołania metody `exec()` w pętli, której wykonywanie zakończy się w chwili gdy nie zostanie odnaleziony żaden łańcuch pasujący do podanego wyrażenia regularnego, to po zakończeniu tej pętli obiekt `RegExp` wciąż będzie zawierać informacje o ostatnim odnalezionym fragmencie, których będzie można użyć w dalszej części skryptu.

Korzystanie z wyrażeń regularnych

Pomimo pozornie skomplikowanego, ukrytego sposobu działania wyrażeń regularnych, język JavaScript zawiera grupę metod, dzięki którym wykonywanie najczęstszych zadań wykorzystujących wyrażenia regularne jest całkiem proste (przy założeniu, że uda Ci się stworzyć dobre wyrażenie regularne). W tym podrozdziale przedstawię sposoby realizacji kilku typów zadań, w których pomocne okazują się wyrażenia regularne.

Czy odnaleziono pasujący fragment?

Wspominałem wcześniej, że do sprawdzenia czy określony łańcuch znaków stanowi część większego łańcucha można wykorzystać metody `string.indexOf()` lub `string.lastIndexOf()`. Jeśli jednak podczas wyszukiwania konieczne jest wykorzystanie możliwości, jakie zapewniają wyrażenia regularne, to można się posłużyć dwiema innymi metodami:

```
obiektWyrReg.test(łańcuchZnaków)
string.search(obiektWyrReg)
```

Pierwsza z nich jest metodą obiektu wyrażenia regularnego, a druga metodą obiektu `String`. Obie te metody wykonują te same zadania i oddziałują na te same obiekty, zwracają jednak inne wartości. Metoda `test()` zwraca wartość logiczną, natomiast metoda `search()` przesuwanie odnalezionego fragmentu od początku łańcucha lub wartość `-1`, jeśli pasujący fragment nie został odnaleziony. Wybór jednej z tych metod zależy od tego, czy w skrypcie potrzebna jest jedynie wartość logiczna określająca czy odnaleziono łańcuch pasujący do wyrażenia regularnego, czy też położenie tego łańcucha.

Przykład przedstawiony na listingu 38.1 przedstawia sposób wykorzystania obu tych metod. Strona zawiera domyślny tekst oraz wyrażenie regularne pozwalające na odnajdywanie pięciocyfrowych liczb. W tym skrypcie obiekt wyrażenia regularnego jest tworzony przy użyciu konstruktora `RegExp`, dlatego też ciągu znaków definiującego wyrażenie regularne nie należy zapisywać pomiędzy znakami ukośników.

Listing 38.1. Wyszukiwanie ciągu pasującego do podanego wyrażenia regularnego

```
<HTML>
<HEAD>
<TITLE>Czy coś pasuje?</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
function findIt(form) {
    var re = new RegExp(form.regexp.value)
    var input = form.main.value
    if (input.search(re) != -1) {
        form.output[0].checked = true
    } else {
        form.output[1].checked = true
    }
}
function locateIt(form) {
    var re = new RegExp(form.regexp.value)
    var input = form.main.value
    form.offset.value = input.search(re)
```

```

}
</SCRIPT>
</HEAD>
<BODY>
<B>Użyj wyrażenia regularnego, aby sprawdzić czy łańcuch znaków zawiera ciąg o podanej
postaci:</B>
<HR>
<FORM>
Podaj tekst, który będzie przeszukiwany:<BR>
<TEXTAREA NAME="main" COLS=40 ROWS=4 WRAP="virtual">
Być może najsłynniejszym kodem pocztowym na świecie jest 90210.
</TEXTAREA><BR>
Podaj wyrażenie regularne:<BR>
<INPUT TYPE="text" NAME="regex" SIZE=30 VALUE="\b\d\d\d\d\b"><P>
<INPUT TYPE="button" VALUE="Czy znaleziono pasujący fragment?"
onClick="findIt(this.form)">
<INPUT TYPE="radio" NAME="output">Tak
<INPUT TYPE="radio" NAME="output">Nie <P>
<INPUT TYPE="button" VALUE="Gdzie?" onClick="locateIt(this.form)">
<INPUT TYPE="text" NAME="offset" SIZE=4><P>
<INPUT TYPE="reset" VALUE="Wyczyść pola">
</FORM>
</BODY>
</HTML>

```

Pobieranie informacji o odnalezionym fragmencie

W kolejnej przykładowej aplikacji, zadanie polega nie tylko na sprawdzeniu czy data wpisana w pojedynczym polu tekstowym została zapisana w odpowiednim formacie, lecz także na pobraniu poszczególnych fragmentów tej daty i przeprowadzeniu na ich podstawie obliczeń mających na celu ustalenie dnia tygodnia. Wyrażenie regularne wykorzystane w tym przykładzie jest dosyć skomplikowane, gdyż dodatkowo, w uproszczony sposób, sprawdza zakres wpisanych wartości, aby upewnić się, że wprowadzona przez użytkownika cyfra oznaczająca miesiąc nie jest większa od 12, a oznaczająca dzień — od 31. Wyrażenie to nie uwzględnia jednak różnic w długości poszczególnych miesięcy. Ale wyrażenie regularne oraz wywoływane metody pobierają poszczególne składowe daty, dzięki czemu można przeprowadzić dodatkowe testy i upewnić się, że użytkownik nie podał błędnej daty 31 września. Należy także pamiętać, że nie jest to jedyny sposób sprawdzania poprawności dat, jaki można wykorzystać w skryptach.

Listing 38.2 przedstawia kod źródłowy strony zawierającej jedno pole tekstowe służące do wprowadzania daty, przycisk, którego kliknięcie powoduje przetworzenie daty oraz kolejne pole tekstowe służące do prezentacji rozbudowanej wersji daty włącznie z dniem tygodnia. Na samym początku funkcji realizującej wszystkie operacje, tworzone są dwie tablice (przy użyciu zapisu literałowego wprowadzonego w języku JavaScript 1.2), zawierające nazwy dni tygodnia oraz miesięcy. Tablice te będą wykorzystywane wyłącznie w przypadku gdy użytkownik poda poprawną datę.

Listing 38.2. Odczytywanie informacji z odszukanego łańcucha znaków

```

<HTML>
<HEAD>
<TITLE>Czy coś pasuje?</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
function extractIt(form) {
    var months =
["stycznia", "lutego", "marca", "kwietnia", "maja", "czerwca", "lipca", "sierpnia", "września",
"października", "listopada", "grudnia"]
    var days =
["niedziela", "poniedziałek", "wtorek", "środa", "czwartek", "piątek", "sobota"]
    var re = /\b(0?[1-9]|[12][0-9]|3[01])[\-\/](1[0-2]|0?[1-9])[\-\/]((19|20)\d{2})/
    var input = form.entry.value
    var matchArray = re.exec(input)
    if (matchArray) {
        var theMonth = months[matchArray[2] - 1] + " "
        var theDate = matchArray[1] + " "
        var theYear = matchArray[3]
        var dateObj = new Date(matchArray[3], matchArray[2]-1, matchArray[1])
        var theDay = days[dateObj.getDay()] + ", "
        form.output.value = theDay + theDate + theMonth + theYear
    } else {
        form.output.value = "Data nie jest poprawna."
    }
}
</SCRIPT>
</HEAD>
<BODY>
<B>Użycie wyrażenia regularnego w celu pobrania informacji o dacie z łańcucha
znaków:</B>
<HR>
<FORM>
Enter a date in the format dd/mm/yyyy or dd-mm-yyyy:<BR>
<INPUT TYPE="text" NAME="entry" SIZE=12><P>
<INPUT TYPE="button" VALUE="Pobierz informacje o dacie"
onClick="extractIt(this.form)"><P>
Wpisana data to:<BR>
<INPUT TYPE="text" NAME="output" SIZE=40><P>
<INPUT TYPE="reset" VALUE="Wyczyść pola">
</FORM>
</BODY>
</HTML>

```

Następnie definiowane jest wyrażenie regularne, z którym będą porównywane dane wpisywane przez użytkownika. Jeśli zadamy sobie trud przeanalizowania tego wyrażenia, przekonamy się, że składa się ono z trzech komponentów, które mogą być od siebie oddzielone łącznikami lub znakami ukośnika (`[\-\/]`). Chcąc odszukać łącznik lub znak ukośnika, podczas definiowania wyrażenia regularnego należy poprzedzić te znaki odwrotnym ukośnikiem. Istotne jest to, że definicje każdego z tych komponentów zostały zapisane w nawiasach, niezbędnych aby wartości tych komponentów zostały zapamiętane w obiektach wyrażenia regularnego.

Oto krótki opis cech, które musi spełniać łańcuch znaków wyszukiwany przez wyrażenie regularne:

- ♦ Łańcuch musi rozpoczynać się od granicy słowa.
- ♦ Łańcuch odpowiadający dacie (dniowi miesiąca) rozpoczynający się od opcjonalnego 0 i zakończony cyfrą od 1 do 9; rozpoczynający się od 1 lub 2 i zakończony cyfrą od 0 do 9 lub rozpoczynający się cyfrą 3 i zakończony cyfrą 0 lub 1.
- ♦ Łącznik lub znak ukośnika.
- ♦ Łańcuch określający miesiąc musi zawierać cyfrę 1 oraz cyfry od 0 do 2 lub musi zawierać cyfrę od 0 do 9 ewentualnie poprzedzoną cyfrą 0.
- ♦ Kolejny łącznik lub znak ukośnika.
- ♦ Łańcuch określający rok, rozpoczynający się od cyfr 19 lub 20 i zakończony dwiema kolejnymi cyframi.

Dodatkowa para nawiasów musi otaczać segment 19|20, aby zapewnić, że jeden z tych dwóch łańcuchów zostanie dołączony do kolejnych dwóch cyfr. Gdyby nawiasy zostały pominięte, to wyrażenie dołączałoby te dwie cyfry wyłączenie do łańcucha 20.

Do zastosowania wyrażenia regularnego zdecydowałem się użyć metody `exec()`, a zwracaną przez nią tablicę zapisuję w zmiennej `matchArray`. W tym miejscu można by także zastosować metodę `string.match()`. Dalsze instrukcje w głównej części funkcji są wykonywane wyłącznie w przypadku, gdy w łańcuchu podanym przez użytkownika odnaleziona zostanie data zgodna z wzorcem (czyli gdy wszystkie warunki zdefiniowane w wyrażeniu regularnym zostaną spełnione).

Nawiasy otaczające każdy z segmentów wyrażenia regularnego instruuja interpreter JavaScriptu, że każdą z odnalezionych wartości należy zapisać w odpowiedniej komórce tablicy `matchArray`. Dzień miesiąca zapisywany jest w komórce `matchArray[1]`, miesiąc w komórce `matchArray[2]`, a rok w komórce `matchArray[3]` (komórka `matchArray[0]` zawiera cały łańcuch znaków pasujący do wyrażenia regularnego). Dzięki takiemu rozwiązaniu skrypt może odczytać wartości poszczególnych elementów daty i na ich podstawie stworzyć jej słowną reprezentację (posługując się dodatkowo tablicami zdefiniowanymi na samym początku funkcji). Skrypt tworzy nawet nowy obiekt daty, który samodzielnie określi potrzebny nam dzień tygodnia. Gdy zostaną już określone wszystkie elementy konieczne do utworzenia daty, skrypt łączy je w jedną całość, po czym wyświetla w polu wynikowym. Jeśli wynik zwrócony przez metodę `exec()` obiektu wyrażenia regularnego wskaże, że podany ciąg nie jest zgodny z wyrażeniem regularnym, to skrypt wyświetla w polu wynikowym stosowny komunikat o błędzie.

Zastępowanie łańcuchów

Aby przedstawić sposób wykorzystania wyrażeń regularnych w operacjach wyszukiwania i zastępowania, postanowiłem posłużyć się aplikacją, która z pewnością przyda się autorom stron prezentujących liczby o bardzo dużych wartościach. W bazach danych duże liczby całkowite są zazwyczaj zapisywane bez dodatkowych znaków ułatwiających ich odczyt (w krajach anglojęzycznych używa się do tego zazwyczaj przecinków, a do oddzielania cyfr na pozycjach dziesiętnych używa się kropki). Jeśli jednak liczba zawiera więcej niż pięć lub sześć cyfr, odczytanie jej wartości staje się dosyć trudne. Jeśli jednak użytkownik dopiero ma wpisać taką dużą liczbę, to dodatkowe znaki mogą mu pomóc w zachowaniu dokładności.

Obiekty wyrażeń regularnych języka JavaScript udostępniają przydatną metodę — `string.replace()` (opisaną w rozdziale 34.) — którą wykorzystamy w tym przykładzie. Metoda ta wymaga podania dwóch argumentów: wyrażenia regularnego używanego do przeszukiwania oraz łańcucha znaków, którym będą zastępowane wszystkie odnalezione fragmenty. Ten drugi łańcuch znaków używany do zastępowania może być określany za pomocą właściwości obiektu `RegExp` — właściwości te podawane są bezpośrednio za wywołaniem metody `exec()`.

Strona z listingu 38.3 pokazuje, że dzięki wykorzystaniu wyrażeń regularnych, kilka wierszy kodu wystarczy do wykonania naprawdę skomplikowanych operacji. Strona zawiera trzy pola. W pierwszym z nich można podać dowolną liczbę. Kliknięcie przycisku *Wstaw przecinki* powoduje wywołanie funkcji `commafy()` zdefiniowanej na naszej przykładowej stronie. Wyniki jej wykonania są wyświetlane w drugim polu tekstowym. Można także podać cyfrę zapisaną z przecinkami w drugim polu tekstowym i kliknąć przycisk *Usuń przecinki*, aby wywołać funkcję `decommafy()`, która wykona operację odwrotną — czyli usunie przecinki z łańcucha znaków.

Listing 38.3. Zastępowanie łańcuchów znaków za pomocą wyrażeń regularnych

```
<HTML>
<HEAD>
<TITLE>Czy coś pasuje?</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
function commafy(form) {
    var re = /(-?\d+)(\d{3})/
    var num = form.entry.value
    while (re.test(num)) {
        num = num.replace(re, "$1,$2")
    }
    form.commaOutput.value = num
}
function decommafy(form) {
    var re = /,/g
    form.plainOutput.value = form.commaOutput.value.replace(re,"")
}
</SCRIPT>
</HEAD>
<BODY>
<B>Wykorzystanie wyrażeń regularnych w celu dodawania/usuwania przecinków z liczb:</B>
<HR>
<FORM>
Podaj dużą liczbę bez przecinków:<BR>
<INPUT TYPE="text" NAME="entry" SIZE=15><P>
<INPUT TYPE="button" VALUE="Wstaw przecinki" onClick="commafy(this.form)"><P>
Liczba z przecinkami:<BR>
<INPUT TYPE="text" NAME="commaOutput" SIZE=20><P>
<INPUT TYPE="button" VALUE="Usuń przecinki" onClick="decommafy(this.form)"><P>
Liczba z usuniętymi przecinkami:<BR>
<INPUT TYPE="text" NAME="plainOutput" SIZE=15><P>
<INPUT TYPE="reset" VALUE="Wyczyść pola">
</FORM>
</BODY>
</HTML>
```

Specyfikacja wyrażenia regularnego dopuszcza zarówno dodatnie, jak i ujemne łańcuchy znaków zawierające same cyfry. Kluczowym elementem z punktu widzenia działania tej aplikacji są nawiasy otaczające dwa segmenty wyrażenia regularnego. Jeden nawias otacza wszystkie znaki, które nie zostały umieszczone w drugim — ciąg trzech cyfr, które liczba musi zawierać. Innymi słowy, wykorzystywane w tym przykładzie wyrażenie regularne działa od tyłu łańcucha znaków, wydzielając z niego trzycyfrowe segmenty i wstawiając przecinek za każdym razem, gdy taki segment zostanie odnaleziony.

Analiza i modyfikacja łańcucha realizowana jest w pętli `while` (w rzeczywistości, istniejący obiekt łańcucha nie jest modyfikowany, zamiast tego podczas każdej modyfikacji tworzony jest nowy obiekt łańcuchowy, który następnie zapisywany jest w istniejącej zmiennej). W tym przypadku, w warunku pętli wykorzystałem metodę `test()`, gdyż nie są nam potrzebne informacje zwracane przez metodę `exec()`. Metoda `test()` modyfikuje właściwości obiektu wyrażenia regularnego oraz obiektu `RegExp` tak samo, jak metoda `exec()`, lecz działa nieco bardziej efektywnie. W efekcie pierwszego wywołania metody `test()`, fragment łańcucha odpowiadający pierwszej części wyrażenia regularnego zostaje zapisany we właściwości `RegExp.$1`, z kolei drugi fragment (jeśli w ogóle jest), zostaje zapisany we właściwości `RegExp.$2`. Warto zwrócić uwagę, że wynik działania metody `exec()` nie jest zapisywany w żadnej zmiennej — w tej aplikacji informacje zwracane przez tę metodę nie są nam potrzebne.

Dochodzimy teraz do kluczowej części skryptu. Wywoływana jest metoda `string.replace()`, dla której punktem wyjścia jest bieżąca wartość łańcucha (`num`). Wyszukiwanym wzorcem jest wyrażenie regularne zdefiniowane na samym początku funkcji. Dosić dziwnie wygląda natomiast łańcuch, który ma zastąpić łańcuch odszukany. Łańcuch ten zastępuje ciąg znaków odnaleziony przez wyrażenie regularne wartością właściwości `RegExp.$1`, połączoną z przecinkiem oraz wartością właściwości `RegExp.$2`. Obiekt `RegExp` nie powinien stanowić części odwołania użytego do wywołania metody `replace()`. Ponieważ wyrażenie regularne obejmuje cały łańcuch znaków `num`, metoda `replace()` w rzeczywistości tworzy ten łańcuch na nowo, budując go z jego poprzedniej zawartości i dodając przecinek przed jego drugim fragmentem (czyli grupą trzech ostatnich cyfr). Każde wywołanie metody `replace()` określa wartość zmiennej `num` przygotowując kolejną iterację pętli `while` i wywołanie metody `test()`.

Realizacja pętli trwa aż do chwili, gdy nie zostanie już odnaleziony ciąg znaków zgodny z wzorcem zdefiniowanym w wyrażeniu regularnym, co oznacza, że w łańcuchu nie ma już żadnych niezależnych trzycyfrowych ciągów znaków. Uzyskane wyniki są następnie zapisywane w drugim polu tekstowym.

Usuwanie przecinków jest jeszcze prostszym zadaniem. Wyrażenie regularne zawiera sam przecinek i modyfikator `g`. Po odnalezieniu tego modyfikatora, metoda `replace()` powtarza cały proces aż do momentu, gdy wszystkie fragmenty łańcucha zgodne z podanym wzorcem zostaną zastąpione. W naszym przypadku łańcuch zastępujący jest po prostu pustym łańcuchem znaków. Więcej przykładów wykorzystania wyrażeń regularnych wraz z obiektami `String` można znaleźć w rozdziale 34., w jego części poświęconej metodom `string.match()`, `string.replace()` oraz `string.split()`.

Obiekt wyrażenia regularnego

Właściwości	Metody
constructor	compile()
global	exec()
ignoreCase	test()
lastIndex	
multiline	
source	

Składnia

Korzystanie z właściwości i metod obiektu wyrażenia regularnego:

```
obiektWyrazeniaRegularnego.właściwość | metoda([argumenty])
```

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓			✓	✓	✓

O obiekcie

Obiekt wyrażenia regularnego jest tworzony podczas wykonywania skryptu. Każdy taki obiekt zawiera swój własny wzorzec oraz wszelkie inne właściwości. Wybór sposobu tworzenia obiektu wyrażenia regularnego zależy od tego, w jaki sposób obiekt ten będzie następnie wykorzystywany w skrypcie.

Gdy tworzy się wyrażenie regularne przy użyciu notacji literałowej (czyli zapisuje je między dwoma znakami ukośnika), wyrażenie to jest automatycznie kompilowane w celu zapewnienia jak największej efektywności działania. To samo dzieje się w przypadku tworzenia obiektów wyrażeń regularnych przy wykorzystaniu konstruktora `RegExp()` i podania w jego wywołaniu łańcucha znaków definiującego wzorzec (wraz z opcjonalnymi modyfikatorami). Za każdym razem, gdy wykorzystywane w skrypcie wyrażenie regularne jest stałe, warto je stworzyć przy użyciu notacji literałowej, jeśli natomiast całe wyrażenie lub jego fragmenty są określane na podstawie informacji podawanych przez użytkownika, to łańcuch znaków definiujący wyrażenie należy przekazać jako argument wywołania konstruktora `RegExp()`. Ze skompilowanego wyrażenia regularnego można w skrypcie skorzystać zawsze wtedy, gdy jest ono gotowe do ponownego wykorzystania. Skompilowane wyrażenia regularne nie są zapisywane na dysku i istnieją wyłącznie do czasu zakończenia działania skryptu wykonywanego na stronie (innymi słowy, są niszczone w momencie usuwania strony z przeglądarki).

Jednak mogą się zdarzyć sytuacje, w których specyfikacja wyrażenia regularnego zmienia się podczas każdej iteracji pętli. Na przykład, jeśli instrukcje wykonywane wewnątrz pętli

`while` modyfikują zawartość wyrażenia regularnego, to wewnątrz pętli należy takie wyrażenie skompilować; poniżej przedstawiłem uproszczony przykład takiego rozwiązania:

```
var srchText = form.search.value
var re = new RegExp() // pusty konstruktor
while (someCondition) {
    re.compile("\\s+" + srchText + "\\s+", "gi")
    instrukcje które mogą zmienić srchText
}
```

Za każdym razem gdy wykonywane są instrukcje umieszczone wewnątrz pętli, tworzony jest nowy wzorzec wyrażenia regularnego (łączony z metaznakami zastępującymi jeden lub więcej znaków odstępu po obu stronach przeszukiwanego tekstu, którego zawartość ulega ciągłym zmianom). Następnie wyrażenie to kompilujemy, tak aby utworzyć efektywny obiekt, którego można używać z dowolnymi skojarzonymi metodami.

Właściwości

constructor

Patr: opis właściwości `string.constructor` podany w rozdziale 34.

global ignoreCase

Wartość: logiczna

Tylko do odczytu

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓					✓

Te dwie właściwości odpowiadają modyfikatorom `g` oraz `i`, które można dodać do wyrażenia regularnego. Ich wartości są określane podczas tworzenia obiektu i można je wyłącznie odczytywać. Właściwości te są od siebie wzajemnie niezależne.

Zagadnienia pokrewne: brak.

lastIndex

Wartość: liczba całkowita

Odczyt i zapis

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓					✓

Właściwość `lastIndex` określa numer indeksu znaku łańcucha głównego, od którego ma się rozpocząć wyszukiwanie fragmentu zgodnego z wzorcem określonym za pomocą wyrażenia regularnego. Bezpośrednio po utworzeniu obiektu wyrażenia regularnego, właściwość ta ma wartość 0, co oznacza, że obiekt ten jeszcze nie był użyty do wyszukiwania, a pierwsze wyszukiwanie, do którego zostanie użyty rozpocznie się domyślnie od samego początku głównego łańcucha znaków.

Jeśli podczas tworzenia wyrażenia regularnego użyto modyfikatora `g` (przeszukiwanie globalne), to po odnalezieniu w łańcuchu głównym fragmentu pasującego do wzorca, wartość tej właściwości jest odpowiednio powiększana — wskazuje ona indeks znaku łańcucha głównego, znajdującego się bezpośrednio za odnalezionym fragmentem (wszystkie znaki odszukanego fragmentu mają niższe indeksy). Po odszukaniu ostatniego fragmentu łańcucha głównego pasującego do wzorca określonego w wyrażeniu regularnym, właściwość tej ponownie przypisywana jest wartość 0. Można także modyfikować przebieg wyszukiwania przez własnoręczne zmienienie wartości tej właściwości. Na przykład, aby wyszukiwanie rozpoczynało się od czwartego znaku łańcucha głównego, należy zmienić wartość właściwości `lastIndex` bezpośrednio po utworzeniu obiektu wyrażenia regularnego:

```
var re = /jakiśWzorzec/
re.lastIndex = 3 // czwarty znak, gdyż pierwszy znak ma indeks zero
```

Zagadnienia pokrewne: właściwość `index` obiektu zwracanego jako wynik wyszukiwania.

multiline

Wartość: logiczna

Tylko do odczytu

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność				✓					✓

Właściwość `multiline` informuje czy wyszukiwanie będzie realizowane w wielu wierszach tekstu stanowiących zawartość głównego łańcucha znaków. Wartość tej właściwości określana jest na podstawie opcjonalnego modyfikatora `m`, którego można użyć podczas tworzenia wyrażenia regularnego. W przeglądarkach NN 4+ właściwość o tej samej nazwie jest także dostępna w statycznym obiekcie `RegExp` (opisanym w następnym podrozdziale).

Zagadnienia pokrewne: właściwość `RegExp.multiline`.

source

Wartość: łańcuch znaków

Tylko do odczytu

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓			✓	✓	✓

Właściwość `source` zawiera jedynie reprezentację łańcuchową wyrażenia użytego do zdefiniowania obiektu. Właściwość ta jest przeznaczona wyłącznie do odczytu.

Zagadnienia pokrewne: brak.

Metody

compile(„wzorzec”, [„g” | „i” | „m”])

Wartość wynikowa: obiekt wyrażenia regularnego

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓			✓	✓	✓

Metody `compile()` należy używać do kompilacji wyrażeń regularnych, których zawartość zmienia się nieustannie podczas działania skryptu. Przykład można znaleźć we wcześniejszym opisie obiektu. W przypadku użycia innego sposobu tworzenia obiektu wyrażenia regularnego (notacji literałowej lub konstruktora `RegExp()`, w którego wywołaniu podaje się wyrażenie regularne), wyrażenie regularne jest kompilowane automatycznie. Modyfikator `m` jest dostępny w przeglądarkach IE 5.5+ oraz NN 6+.

Zagadnienia pokrewne: brak.

exec(„łańcuchZnaków”)

Wartość wynikowa: tablica z informacjami o odnalezionym ciągu lub `null`.

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓			✓	✓	✓

Metoda `match()` sprawdza czy w łańcuchu przekazany jako argument jej wywołania znajduje się przynajmniej jeden fragment zgodny z wzorcem określonym w wyrażeniu regularnym. Działanie tej metody przypomina działanie metody `string.match()` (choć metoda `match()` ma znacznie większe możliwości jeśli chodzi o wykonywanie wyszukiwań o charakterze globalnym). Zazwyczaj metoda `exec()` jest wywoływana bezpośrednio po utworzeniu obiektu wyrażenia regularnego, jak w poniższym przykładzie:

```
var re = /jakiśWzorzec/
var matchArray = exec("jakiśŁańcuchZnaków")
```

Wywołanie metody `exec()` ma wiele następstw. Właściwości obiektu wyrażenia regularnego oraz statycznego obiektu `RegExp`, są aktualizowane w zależności od tego czy udało się odnaleźć fragment pasujący do wzorca, czy też nie. Metoda ta zwraca także tablicę, która dostarcza dodatkowych informacji na temat przeprowadzonej operacji. Właściwości tej tablicy przedstawione zostały w tabeli 38.4.

Tabela 38.4. Właściwości tablicy opisującej odszukany ciąg

Właściwość	Opis
index	Indeks (liczony od zera) określający położenie początku odnalezionego fragmentu w głównym łańcuchu znaków.
input	Cały oryginalny łańcuch znaków.
[0]	Odnaleziony łańcuch znaków.
[1]...[n]	Ciągi znaków odpowiadające wartościom fragmentów wyrażenia regularnego zapisanym w nawiasach.

Niektóre z właściwości tej zwracanej tablicy odpowiadają właściwościom obiektu `RegExp`. Jednak zapisywanie ich w obiekcie wyrażenia regularnego ma tę zaletę, iż można je bezpiecznie przechować, natomiast wartości właściwości obiektu `RegExp` mogą się zmienić w każdej chwili, na skutek innego wywołania jakiejś metody wyrażenia regularnego. Elementy wspólne dla obu obiektów to: właściwość `[0]` (odpowiadająca właściwości `RegExp.lastMatch`) oraz właściwości `[1]` do `[n]` (pierwsze dziewięć z nich odpowiada właściwościom `RegExp.$1` do `RegExp.$9`). Pomimo że obiekt `RegExp` zapamiętuje tylko dziewięć wartości składowych wyrażenia umieszczonych w nawiasach, w zwróconej tablicy zapamiętywanych jest ich tyle, ile potrzeba do uwzględnienia wszystkich składowych wyrażenia zapisanych w nawiasach.

Jeśli w przeszukiwanym łańcuchu znaków nie uda się odnaleźć fragmentu pasującego do wyrażenia regularnego, metoda `exec()` zwraca wartość `null`. Przykład praktycznego wykorzystania tej metody został przedstawiony na listingu 38.2. Metodę `exec()` można także wywoływać w uproszczony sposób, wymaga on potraktowania obiektu wyrażenia regularnego jako metody, jak w poniższym przykładzie:

```
var re = /jakiśWzorzec/
var matchArray = re("jakiśŁańcuchZnaków")
```

Zagadnienia pokrewne: metoda `string.match()`.

test(„łańcuchZnaków”)

Wartość wynikowa: logiczna

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓			✓	✓	✓

Najbardziej efektywnym sposobem sprawdzania czy w łańcuchu znaków znajduje się fragment pasujący do wyrażenia regularnego, jest wykorzystanie metody `test()`. Metoda ta zwraca wartość `true`, jeśli pasujący fragment został odnaleziony, lub wartość `false` w przeciwnym razie. Jeśli informacja zwracana przez tę metodę nie jest wystarczająca, można dodatkowo wywołać metodę `string.search()` zwracającą indeks miejsca, w którym zaczyna się odnaleziony fragment. Przykład wykorzystania tej metody został przedstawiony na listingu 38.1.

Zagadnienia pokrewne: metoda `string.search()`.

Obiekt RegExp

Właściwości	Metody
input	
lastMatch	
lastParen	
leftContext	
multiline	
prototype	
rightContext	
\$1, ... \$9	

Składnia

Odwoływanie się do właściwości obiektu RegExp:

`RegExp.właściwość`

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓			✓	✓	✓

O obiekcie

Począwszy od Netscape Navigator 4 oraz Internet Explorera 4, w każdym oknie przeglądarki lub ramce tworzona jest kopia obiektu RegExp. Obiekt ten nadzoruje działania wszystkich metod wykorzystujących wyrażenia regularne (włącznie z kilkoma metodami obiektu String). Właściwości tego obiektu można wykorzystywać w skryptach nie tylko w tradycyjny sposób, dodatkowo można ich także używać jako argumentów wywołania metody `string.replace()` (patrz listing 38.3).

Fakt, że dysponujemy tylko jednym obiektem RegExp obsługującym wszystkie operacje wykonywane na wyrażeniach regularnych we wszystkich skryptach działających w danym dokumencie, wymusza zachowanie szczególnej ostrożności podczas odwoływania się i modyfikacji jego właściwości. Koniecznie należy upewnić się, że obiekt RegExp nie został zmodyfikowany przez wywołanie innej metody. Wywołanie jakiegokolwiek metody wykorzystującej wyrażenia regularne może bowiem doprowadzić do aktualizacji wartości wielu właściwości tego obiektu. Właśnie z tego względu warto zastanowić się nad użyciem właściwości tablicy zwracanej przez większość metod wyrażen regularnych, zamiast właściwości obiektu RegExp. Właściwości tablicy przypisywane są bowiem do obiektu zawierającego konkretne wyrażenia regularne i nie zmieniają się nawet po użyciu w tym samym skrypcie innego wyrażenia regularnego. Z kolei właściwości obiektu RegExp odzwierciedlają ostatnią wykonaną operację wykorzystującą wyrażenia regularne, niezależnie od tego, jaki obiekt wyrażenia regularnego został użyty w tej operacji.

W opisach podanych w dalszej części rozdziału wykorzystywane są długie, charakterystyczne dla JavaScriptu nazwy właściwości. Każdą z tych nazw można jednak zastąpić krótszym odpowiednikiem, charakterystycznym dla skryptów pisanych w języku Perl. Skrócone nazwy właściwości można, na przykład, wykorzystywać w wywołaniach metody `string.replace()`.

Właściwości

input

Wartość: łańcuch znaków

Odczyt i zapis

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓					✓

Właściwość `RegExp.input` zawiera główny łańcuch znaków, w którym wyszukiwane są fragmenty pasujące do wzorca podanego w wyrażeniu regularnym. We wszystkich przykładach przedstawionych we wcześniejszej części rozdziału, właściwość ta miała wartość `null`. Dzieje się tak w sytuacji, gdy łańcuch główny przekazywany jest jako argument wywołania metody skojarzonej z wyrażeniem regularnym.

Większość „tekstowych” obiektów dokumentu z obiektem `RegExp` łączy niewidoczne związki. Otóż, jeśli obiekt pola tekstowego, elementu `TEXTAREA`, `SELECT` lub połączenia zawiera procedurę obsługi zdarzeń wykorzystującą wyrażenie regularne, to we właściwości `RegExp.input` zapisywana jest wartość tekstowa pobrana z danego obiektu. Do procedury obsługi zdarzeń, ani do funkcji wywoływanej przez tę procedurę nie trzeba przekazywać żadnych argumentów. W przypadku obiektów pól tekstowych oraz elementów `TEXTAREA`, we właściwości `input` zapisywana jest zawartość obiektu; w przypadku elementów `SELECT` we właściwości zapisywany jest tekst (nie wartość) wybranej opcji; a w przypadku połączeń — wyróżniony w przeglądarce tekst, skojarzony z połączeniem (dostępny także jako wartość właściwości `text` obiektu reprezentującego połączenie).

To automatyczne określanie wartości właściwości `RegExp.input` może uprościć skrypt. Dzięki niemu można wywołać dowolną metodę wyrażenia regularnego bez konieczności podawania głównego łańcucha znaków jako argumentu jej wywołania. Jeśli argument określający łańcuch główny nie zostanie jawnie podany, JavaScript wykorzysta zamiast niego wartość właściwości `RegExp.input`. Wartość tej właściwości można także określić w każdej chwili podczas wykonywania skryptu. Skrótowe odwołanie do tej właściwości ma postać: `$_` (znak dolara i znak podkreślenia).

Zagadnienia pokrewne: właściwość `input` tablicy opisującej odszukany ciąg.

multiline

Wartość: logiczna

Zapis i odczyt

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓					

Właściwość `RegExp.multiline` określa czy przeszukiwanie ma obejmować więcej niż jeden wiersz łańcucha głównego. Jeśli funkcja wykorzystująca wyrażenie regularne zostanie wywołana przez jakąś procedurę obsługi zdarzeń obiektu `TEXTAREA`, to właściwość ta automatycznie przyjmuje wartość `true`. Wartość tej właściwości można także określić samodzielnie w dowolnej chwili. Skrócone odwołanie do tej właściwości ma postać: `$*`. Ta wartość właściwości `multiline` (w odróżnieniu od analogicznej właściwości obiektu wyrażenia regularnego) nie została uwzględniona w specyfikacji ECMA-262 i jest dostępna wyłącznie w przeglądarkach NN 4+.

Zagadnienia pokrewne: właściwość `multiline` obiektu wyrażenia regularnego.

lastMatch

Wartość: łańcuch znaków

Tylko do odczytu

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓					✓

Po wykonaniu dowolnej metody związanej z wyrażeniem regularnym, dowolny fragment głównego łańcucha znaków pasujący do wzorca podanego w wyrażeniu jest zapisywany we właściwości `RegExp.lastMatch`. Wartość ta jest także zapisywana we właściwości `[0]` tablicy zwracanej przez metody `exec()` oraz `string.match()`. Skrócone odwołanie do tej właściwości ma postać: `$&`.

Zagadnienia pokrewne: właściwość `[0]` tablicy opisującej odnaleziony ciąg.

lastParen

Wartość: łańcuch znaków

Tylko do odczytu

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓					✓

Gdy wyrażenie regularne zawiera wiele komponentów zapisanych w nawiasach, obiekt RegExp przechowuje łańcuchy znaków odpowiadające tym komponentom we właściwościach \$1 do \$9. Wartość ostatniego dopasowanego komponentu zapisanego w nawiasach, można odczytać przy użyciu właściwości RegExp.lastParen. Właściwość ta jest przeznaczona wyłącznie do odczytu. Skrócona forma zapisu tej właściwości to: \$+.

Zagadnienia pokrewne: właściwości RegExp.\$1 do \$9.

leftContext rightContext

Wartość: łańcuch znaków

Tylko do odczytu

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓					✓

Gdy w wyniku wykonania dowolnej z metod zostanie odszukany fragment łańcucha głównego pasujący do wzorca zdefiniowanego w wyrażeniu regularnym, do obiektu RegExp przekazywane są niezwykle ważne informacje o sąsiedztwie odszukanego ciągu. Właściwość leftContext zawiera fragment łańcucha głównego znajdujący się z lewej strony odszukanego fragmentu (jednak bez tego odszukanego ciągu). Należy pamiętać, że łańcuch stanowiący wartość właściwości leftContext zaczyna się w miejscu, w którym rozpoczęło się ostatnie wyszukiwanie. Z tego względu, podczas każdego kolejnego przeszukania tego samego łańcucha głównego przy użyciu tego samego wyrażenia regularnego, uzyskujemy całkowicie odmienne wartości właściwości leftContext.

Właściwość rightContext zawiera łańcuch znaków rozpoczynający się bezpośrednio za odszukanym fragmentem i sięgającym aż do końca głównego łańcucha znaków. Kolejne wywołania metod operujących na tym samym łańcuchu głównym i wykorzystujących to samo wyrażenie regularne, powodują oczywiście sukcesywne skracanie łańcucha stanowiącego wartość tej właściwości. W chwili gdy w łańcuchu głównym nie zostaną już odnalezione żadne fragmenty pasujące do wzorca w wyrażeniu regularnym, właściwość rightContext przyjmuje wartość null. Skrócowa nazwa właściwości leftContext ma postać \$`, a właściwości rightContext — \$'.

Zagadnienia pokrewne: brak.

prototype

Patrz: opis właściwości *string*.prototype podany w rozdziale 34.

\$1 ... \$9

Wartość: łańcuch znaków

Tylko do odczytu

	NN 2	NN 3	NN 4	NN 6	IE 3/J1	IE 3/J2	IE 4	IE 5	IE 5.5
Zgodność			✓	✓			✓	✓	✓

W momencie wywołania metody wyrażenia regularnego, ciągu odpowiadające fragmentom wyrażenia umieszczonym w nawiasach są zapisywane w dziewięciu właściwościach obiektu `RegExp` przeznaczonych właśnie do tego (i określanych jako odwołania wsteczne). Te same wartości (oraz wszystkie pozostałe, które nie zostały zapisane w obiekcie `RegExp`) zostają zapisane w tablicy zwracanej przez metody `exec()` oraz `string.match()`. Poszczególne wartości są zapisywane zgodnie z kolejnością pojawiania się lewych nawiasów w wyrażeniu regularnym, bez względu na zagnieżdżanie pozostałych elementów wyrażenia.

Odwołań wstecznych można używać bezpośrednio w drugim argumencie wywołania metody `string.replace()`, bez konieczności poprzedzania ich odwołaniem do obiektu `RegExp`. Optymalnym rozwiązaniem jest umieszczenie w nawiasach wszystkich komponentów, które w jakikolwiek sposób mają zostać zmodyfikowane lub zastąpione. Na przykład, przedstawiona poniżej funkcja zamienia kolejność zapisu nazwiska i imienia:

```
function swapEM() {
    var re = /(\w+)\.s*(\w+)/
    var input = "Lincoln, Abraham"
    return input.replace(re, "$2 $1")
}
```

W metodzie `replace()` drugi komponent zapisany w nawiasach (imię) jest przenoszony na początek wynikowego łańcucha znaków, do niego dołączany jest odstęp oraz drugi komponent. Przecinek występujący w początkowym łańcuchu znaków jest pomijany. Skrócone nazwy właściwości można ze sobą dowolnie łączyć, a nawet używać ich wielokrotnie, jeśli oczywiście jest to potrzebne.

Zagadnienia pokrewne: właściwości od `[1]` do `[n]` tablicy zawierającej informacje o odszukanym ciągu.