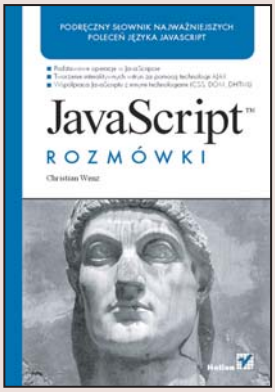
JavaScript. Rozmówki



Autor: Christian Wenz
Tłumaczenie: Leszek Sagalara
ISBN: 978-83-246-0874-4
Tytuł oryginału: JavaScript Phrasebook
Format: B5, stron: 288

Podręczny słownik najważniejszych poleceń języka JavaScript

- Podstawowe operacje w JavaScript
- Tworzenie interaktywnych witryn za pomocą technologii AJAX
- Współpraca JavaScript z innymi technologiami (CSS, DOM, DHTML)

Od pewnego czasu język JavaScript przeżywa odrodzenie i zyskuje zasłużoną popularność. Wynika to przede wszystkim z rozwoju technologii AJAX, która umożliwia tworzenie reaktywnych witryn internetowych i bazuje właśnie na tym języku. JavaScript pozwala na przeprowadzanie wielu przydatnych operacji niewykonalnych w zwykłym języku HTML, na przykład kontrolowane otwieranie nowych okien, sprawdzanie poprawności danych w formularzach czy dynamiczne modyfikowanie rysunków. Łatwa składnia oraz szeroki zakres wykonywanych zadań sprawiły, że JavaScript jest jednym z najpopularniejszych języków skryptowych wykonywanych po stronie przeglądarki.

„JavaScript. Rozmówki" to zwięzły i przystępny przewodnik omawiający popularne problemy, pojawiające się w czasie pracy z tym językiem, oraz podsuwający szybkie i skuteczne ich rozwiązania. Korzystając z niego, nauczysz się używać zarówno podstawowych, jak i zaawansowanych funkcji tego języka. Dowiesz się, jak do stron dodawać grafikę, animacje, treści multimedialne oraz jak zastosować JavaScript do tworzenia witryn za pomocą technologii AJAX. Poznasz możliwości współpracy języka JavaScript z innymi technologiami, takimi jak CSS, DOM czy DHTML, a także opanujesz manipulowanie cookies, formularzami, oknami czy ramkami.

- Podstawowe operacje w JavaScript
- Dodawanie grafiki, animacji i treści multimedialnych
- Praca ze stylami CSS
- Obsługa cookies
- JavaScript w DHTML
- Manipulowanie modelem DOM
- Obiekty i zdarzenia w JavaScript
- Podstawy technologii AJAX
- Obsługa formularzy
- Korzystanie z usług Web Service

Dzięki zwięzłym rozmówkom w błyskawicznym tempie usystematyzujesz swoją wiedzę o języku JavaScript

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

Spis treści

2

Podstawowe zwroty

Istnieją pewne powtarzające się zadania JavaScript, które musimy przeprowadzać niemal codziennie. Stanowią one podstawę wielu aplikacji JavaScript i nie pasują do żadnej określonej kategorii. Ten rozdział zawiera zbiór często spotykanych problemów oraz ich rozwiązań.

Wykrywanie przeglądarki

```
window.alert(navigator.appName);
```

Mimo iż obecnie implementacje JavaScriptu w przeglądarkach są ze sobą dość zgodne (zwłaszcza gdy porównamy to z sytuacją, która panowała w czasie wojny przeglądarek pod koniec lat 90.), to wykrywanie przeglądarki stanowi nadal ważny element warsztatu programisty JavaScript.

Obiekt JavaScriptu `navigator` dostarcza informacji na temat przeglądarki. Najbardziej przydatna, choć czasem trudna do analizy, jest jego właściwość `userAgent`, zawierająca

łańcuch tekstowy identyfikujący przeglądarkę, który jest przesyłany przy każdym żądaniu w nagłówku HTTP `User-Agent`.

Do określenia rodzaju przeglądarki wystarczy sama właściwość `appName`, co przedstawia kod powyżej. Tabela 2.1 podaje wartości `appName` dla najważniejszych przeglądarek.

Tabela 2.1. Wartości appName dla różnych przeglądarek

Przeglądarka	appName
Internet Explorer	`Microsoft Internet Explorer`
Przeglądarki Mozilla	`Netscape`
Konqueror (KDE)	`Konqueror`
Apple Safari	`Netscape`
Opera	`Opera`

Jak widać, przeglądarka Safari podaje niepoprawną nazwę. Aby zniwelować ten efekt, możemy odszukać rodzaj przeglądarki w `navigator.userAgent`. Ponieważ przeglądarka Opera potrafi identyfikować się jako inna przeglądarka (choć w `navigator.userAgent` wciąż występuje ciąg `"Opera"`), należy ją sprawdzić w pierwszej kolejności.

```
<script language="JavaScript"
➥type="text/javascript">
var uA = navigator.userAgent;
var browserType = "unknown";
if (uA.indexOf("Opera") > -1) {
  browserType = "Opera";
} else if (uA.indexOf("Safari") > -1) {
  browserType = "Safari";
} else if (uA.indexOf("Konqueror") > -1) {
  browserType = "Konqueror";
} else if (uA.indexOf("Gecko") > -1) {
```

```
  browserType = "Mozilla";
} else if (uA.indexOf("MSIE") > -1) {
  browserType = "Internet Explorer";
}
window.alert(browserType);
</script>
```

Wykrywanie przeglądarki (browser.html)

Przy odrobinie wysiłku można rozbudować ten skrypt o możliwość wykrywania pochodnych Mozilli (Firefox, Epiphany, Galeon, Camino, SeaMonkey itp.).

Wykrywanie numeru wersji przeglądarki

Numer wersji danej przeglądarki można wykryć na kilka sposobów. W większości wypadków jest on podany w `navigator.userAgent`, co może wyglądać tak:

```
Mozilla/5.0 (Windows; U; Windows NT 5.1; en;
➥rv:1.8.0.3) Gecko/20060426 Firefox 1.5.0.3
➥Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.4) Gecko/20030619 Netscape/7.1 (ax)
➥Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
➥SV1; .NET CLR 1.0.3705; .NET CLR 1.1.4322; .NET
➥CLR 2.0.50727)
Mozilla/5.0 (compatible; Konqueror/3.4; FreeBSD)
➥KHTML/3.4.2 (like Gecko)
Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en)
➥AppleWebKit/418 (KHTML, like Gecko) Safari/417.9.3
Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en)
➥AppleWebKit/312.8 (KHTML, like Gecko) Safari/312.6
Opera/9.00 (Windows NT 5.1; U; en)
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
➥en) Opera 9.00
```

Jak widzimy, w zależności od rodzaju przeglądarki numer wersji jest ukryty gdzieś w obrębie wartości `navigator.UserAgent`. Dlatego też rozpoznawanie wszystkich możliwych przeglądarek i śledzenie nowych metod jest zadaniem niezwykle żmudnym. Istnieją jednak serwisy internetowe oferujące całkiem przyzwoite wykrywanie przeglądarek. Dokładna dokumentacja i kod znajdują się w witrynach:

- *http://www.webreference.com/tools/browser/*
  ➥javascript.html
- *http://gemal.dk/browserspy/basic.html*

Sprawdzanie możliwości przeglądarki

```
if (document.getElementById) {
  //…
```

Jak widać z poprzedniego przykładu, poleganie na numerze wersji przeglądarki nie tylko jest trudne, ale także nie wiadomo, czy zadziała w przypadku przyszłych wersji. Znacznie lepszą metodą określenia, czy dana przeglądarka obsługuje funkcje wymagane przez naszą aplikację, jest sprawdzenie obsługi obiektów specjalnych.

Na przykład, aby użyć DOM (patrz rozdział 5., „DOM i DHTML"), możemy zastosować przedstawiony wyżej kod. Jeśli zaimplementowana jest metoda `getElementById()`, wówczas `document.getElementById` (bez nawia-

sów) zwróci odwołanie do funkcji. Przy zastosowaniu wewnątrz warunku da wartość `true`, co spowoduje wykonanie powiązanego kodu.

Inny przykład: dla pewnych aplikacji Internet Explorer obsługuje obiekty ActiveX, na przykład jeśli chodzi o obsługę XML-a. Jednak ActiveX jest obsługiwany tylko przez Internet Explorera w systemie Windows, ale już nie w systemach Macintosh. Tak więc samo sprawdzenie obecności przeglądarki Internet Explorer spowoduje problemy u użytkowników systemów Macintosh. Ominiemy ten problem, sprawdzając obsługę ActiveX:

```
if (window.ActiveXObject) {
  //…
}
```

Zapobieganie buforowaniu

```
document.write("<img src=\"image.png?" +
➥ Math.random() + "\" />");
```

Korzystając z nagłówków po stronie serwera, możemy uniknąć buforowania w pamięci podręcznej zawartości tworzonej dynamicznie — tak plików graficznych, jak i stron HTML. Nie jest to jednak idealne rozwiązanie, ponieważ niektóre przeglądarki lub serwery proxy mogą ignorować te ustawienia. W takim przypadku rozwiązaniem jest dodanie do adresu URL losowego łańcucha przez użycie `Math.random()`, co zwróci losową liczbę z zakresu od 0 do 1, np. `0.1296601696732852`. Dodanie takiego

łańcucha do adresu pliku graficznego zwykle nie zmienia danych przesyłanych z serwera, natomiast dla przeglądarki stanowi zupełnie nowe żądanie. Dzięki temu takie pliki graficzne (lub inne dane) nie będą pobierane z pamięci podręcznej.

Przekierowania

```
location.href = "newPage.html";
```

Właściwość `location.href` daje dostęp do odczytu i zapisu adresu URL aktualnej strony. W efekcie nadanie `location. href` innej wartości wywoła przekierowanie przeglądarki, która wczyta nową stronę, co przedstawiono na powyższym przykładzie.

WSKAZÓWKA

Można to również wykonać za pomocą HTML-a:

```
<meta http-equiv="Refresh" content="X; URL=Y" />
```

Symbol `X` oznacza czas (w sekundach), po którym rozpocznie się wczytywanie nowej strony, a `Y` wskazuje adres URL nowej strony.

Poprzednia strona zostanie umieszczona w historii przeglądarki. Jeśli jednak chcemy zastąpić w historii starą stronę (aby zmienić działanie przycisku *Wstecz*), możemy zastosować metodę `location.replace()`:

```
location.replace("newPage.html");
```

Odświeżanie strony

```
location.reload();
```

Metoda `reload()` obiektu `location` powoduje odświeżenie aktualnej strony, co jest odpowiednikiem `location.href = location.href`. Podając parametr `true`, możemy wyłączyć działanie pamięci podręcznej, co spowoduje „twarde" odświeżenie strony z serwera. Nie jest to jednak idealne rozwiązanie, gdyż po drodze może działać serwer proxy przechowujący kopię żądanej strony. Możemy więc zamiast tego użyć techniki przedstawionej w podrozdziale „Zapobieganie buforowaniu":

```
location.search = "?" + Math.random();
```

Zmieni to ciąg zapytania (`location.search`) bieżącej strony, skutecznie odświeżając adres URL przez `reload()`.

Tworzenie losowej liczby

```
var rand = min + Math.floor((max – min + 1) *
➥Math.random());
```

Metoda `random` obiektu `Math` oblicza pseudolosową liczbę z zakresu od 0 do 1 (z wyłączeniem tych liczb). Zazwyczaj jednak interesuje nas liczba z zakresu np. od 1 do 10. Można to osiągnąć za pomocą prostych obliczeń matematycznych. Na przykład pomnożenie wyniku `Math.➥random()` przez 10 da liczbę z zakresu od 0 do 10 (z wy-

łączeniem tych liczb). Po zaokrągleniu tej wartości otrzymamy liczbę całkowitą z zakresu od 0 do 9 (włącznie). Dodanie 1 da nam liczbę z zakresu od 1 do 10.

Powyżej zamieszczono ogólną postać kodu, który tworzy liczbę z zakresu między `min` a `max`.

Data i czas

```
var d = new Date();
var dmy = d.getDate() + "/" + (d.getMonth()+1) + "/"
➥+ d.getFullYear();
```

W JavaScripcie obiekt `Date` podaje aktualną datę i daje możliwość przeprowadzenia pewnych obliczeń w stosunku do daty (w przypadku wartości epoki otrzymamy liczbę milisekund, które upłynęły od 1 stycznia 1970 r.). Tabela 2.2 zawiera najważniejsze metody klasy `Date`. Powyższy kod tworzy datę w formacie `dzień/miesiąc/rok`.

Tabela 2.2. Niektóre właściwości Date

Metoda	Opis
`getDate()`	Dzień miesiąca
`getFullYear()`	Rok zapisany w postaci czterocyfrowej
`getHours()`	Godzina
`getMinutes()`	Minuty
`getMonth()`	Miesiąc minus 1 (!)
`getSeconds()`	Sekundy
`getTime()`	Wartość epoki
`toString()`	Przekształcenie w łańcuch znaków
`toUTCString()`	Przekształcenie w łańcuch znaków przy użyciu czasu uniwersalnego (UTC)

Wyrażenia regularne

Wyrażenia regularne to — mówiąc w uproszczeniu — wzorce, do których pasują pewne ciągi znaków. W wyrażeniu regularnym wzorce zawierają łańcuch, który jest wyszukiwany w większym łańcuchu. Można to jednak zrobić (szybciej) za pomocą `indexOf()`. Zaletą wyrażeń regularnych jest możliwość skorzystania z pewnych specjalnych opcji, takich jak symbole wieloznaczne. Tabela 2.3 przedstawia niektóre znaki specjalne i ich znaczenie.

Tabela 2.3. Znaki specjalne w wyrażeniach regularnych

Znak specjalny	Opis	Przykład
^	Początek łańcucha	^a oznacza łańcuch rozpoczynający się na literę a
$	Koniec łańcucha	a$ oznacza łańcuch zakończony literą a
?	0 lub 1 raz (odnosi się do liczby wystąpień znaku lub wyrażenia poprzedzającego)	ab? oznacza a lub ab
*	0 lub więcej razy (odnosi się do znaku lub wyrażenia poprzedzającego)	ab* oznacza a lub ab lub abb lub…
+	1 lub więcej razy (odnosi się do znaku lub wyrażenia poprzedzającego)	ab+ oznacza ab lub abb lub abbb lub…

Znak specjalny	Opis	Przykład
`[...]`	Zestaw znaków do wyboru	`PHP[45]` oznacza `PHP4` lub `PHP5`
`-` (w obrębie nawiasów kwadratowych)	Ciąg wartości	`ECMAScript [3-5]` oznacza `ECMAScript 3` lub `ECMAScript 4` lub `ECMAScript 5`
`^` (w obrębie nawiasów kwadratowych)	Do wzorca pasuje wszystko oprócz podanych znaków	`[^A-C]` oznacza `D` lub `E` lub `F` lub...
`|`	Wzorce alternatywne	`ECMAScript 3|ECMAScript 4` oznacza `ECMAScript 3` lub `ECMAScript 4`, podobnie jak `ECMAScript (3|4)`
`(...)`	Definiuje wzorzec składowy (ang. *Subpattern)*	`(a)(b)` oznacza `ab`, ale z wykorzystaniem dwóch wzorców składowych (`a` i `b`)
`.`	Dowolny znak	`.` oznacza `a`, `b`, `c`, `0`, `1`, `2`, `$`, `^`...
`{min, max}`	Minimalna i maksymalna liczba wystąpień; pominięcie `min` lub `max` oznacza odpowiednio `0` lub nieskończoność (`infinite`)	`a{1,3}` oznacza `a`, `aa` lub `aaa`. `a{,3}` oznacza pusty łańcuch, `a`, `aa` lub `aaa`. `a{1,}` oznacza `a`, `aa`, `aaa`...
`\`	Wprowadzenie następnego znaku	`\.` oznacza `.`

Występują też inne znaki i wyrażenia specjalne, na przykład (`\d`) oznacza znak będący cyfrą.

Wyszukiwanie przy użyciu wyrażeń regularnych

```
zip.test("Indianapolis, IN 46240");
```

W JavaScripcie możemy zdefiniować wyrażenie regularne na dwa sposoby:

- `var zip = new RegEx("\\d{5}");`
- `var zip = /\d{5}/;`

Te dwa podejścia nie różnią się funkcjonalnie, trzeba jedynie wziąć pod uwagę znak ucieczki; następnie za pomocą metody `test()` sprawdzimy, czy łańcuch zawiera wyrażenie regularne:

```
var found = zip.test("Indianapolis, IN 46240");
//true
```

Jeśli interesuje nas wynik dopasowania, użyjemy funkcji `exec()`. Metoda ta zwraca tablicę, której pierwszym elementem jest całe dopasowanie, a wszystkie pozostałe elementy to wyszukiwania podrzędne (jeśli w wyrażeniu regularnym zastosowano nawiasy).

```
var matches = zip.exec("Indianapolis, IN 46240");
// ["46240"]
```

WSKAZÓWKA

Metoda `match()` zwraca wszystkie trafienia, natomiast `exec()` tylko bieżące (zazwyczaj pierwsze). Jeżeli jednak będziemy wielokrotnie wywoływać `exec()`, otrzymamy wszystkie trafienia.

Zamiana tekstu

```
var address = /(\w+), ([A-Z]{2}) (\d{5})/;
var sams = "Indianapolis, IN 46240";
var result = sams.replace(address, "$3 $1, $2");
```

Metoda `replace()`, obsługująca każdy łańcuch JavaScript, służy do zamiany tekstu. Przeprowadza ona wyszukiwanie za pomocą wyrażenia regularnego i zastępuje pasujący łańcuch innym. W obrębie łańcucha zastępującego można stosować odwołania do dopasowań podrzędnych. `0$` wskazuje na pierwsze dopasowanie, `$1` odwołuje się do pierwszego dopasowania podrzędnego (wewnątrz nawiasów), `$2` określa drugie dopasowanie itd. Zamieszczony powyżej kod przeszukuje podane elementy (miasto, stan i kod pocztowy), a następnie je przegrupowuje. Wynikiem jest `"46240 Indianapolis, IN"`.

Nawigacja po historii przeglądarki

```
window.history.back();
window.history.forward();
```

Za historię przeglądarki odpowiada obiekt `history` (właściwość obiektu `window`), który zawiera listę stron WWW odwiedzonych przed aktualną stroną (a jeśli jest to dostępne — także po). I choć z technicznego punktu widzenia możliwe jest przejście o kilka elementów w obrębie historii, to ze względów bezpieczeństwa możliwy jest tylko

jeden sposób: przejście o jedną stronę wstecz i o jedną stronę dalej. Odpowiadają za to poniższe dwie metody:

- `back()` — powoduje przejście do poprzedniej strony w historii (podobnie jak przycisk *Wstecz*);
- `forward()` — powoduje przejście do następnej strony w historii (podobnie jak przycisk *Dalej*).

Wyświetlanie daty modyfikacji strony

```
document.write(document.lastModified);
```

Za każdym razem, gdy serwer WWW wysyła jakieś zasoby do klienta, przesyła również datę ostatniej modyfikacji dokumentu. Zwykle serwer WWW uzyskuje tę informację z systemu plików, ale ten nagłówek można zmodyfikować lub po prostu go nie wysyłać. Tak czy inaczej możemy użyć tej informacji, na przykład w sposób podany w powyższym przykładzie. Dzięki temu otrzymamy mniej lub bardziej realistyczną informację o dacie modyfikacji strony.

Pobieranie parametrów GET

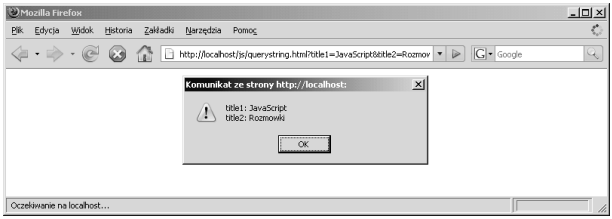
```
var ls = location.search.substring(1);
var namevalue = ls.split("&");
```

Pobieranie parametrów GET

Zazwyczaj informacja GET określana jest po stronie serwera, ale JavaScript może uzyskać do niej dostęp za pomocą właściwości `location.search`. Jednak dane występują tu w postaci par nazwa-wartość. Poniższy kod odczytuje te dane za pomocą metody JavaScript `split()`. Ukaże się wówczas wynikowa tablica przyporządkowująca, aby potwierdzić, że wszystko działa prawidłowo. Wynik działania kodu przedstawia rysunek 2.1.

```
<script language="JavaScript"
➥type="text/javascript">
var getdata = [];
if (location.search.length > 1) {
  var ls = location.search.substring(1);
  var namevalue = ls.split("&");
  for (var i=0; i<namevalue.length; i++) {
    var data = namevalue[i].split("=");
    getdata[data[0]] = data[1];
  }
}
var s = "";
for (var el in getdata) {
  s += el + ": " + getdata… + "\n";
}
alert(s);
</script>
```

Analiza ciągu parametrów (querystring.html)



Rysunek 2.1. Analiza i wyświetlenie danych z ciągu parametrów

Prośba o potwierdzenie przez użytkownika

```
<a href="anyPage.html" onclick="return
➥window.confirm('Czy na pewno tego
➥chcesz?');">Kliknij tutaj</a>
```

JavaScript zapewnia jedynie ograniczoną obsługę okien modalnych. Najczęściej stosowana jest metoda `window.alert()`, ale są też inne możliwości. Jeśli użyjemy `window.confirm()`, wyświetlone zostanie okno z przyciskami *Tak* i *Nie*. Kliknięcie przycisku *Tak* zwróci `true`; w przeciwnym razie wartością wynikową będzie `false`. W powyższym kodzie (plik *confirm.html*) metoda ta pełni rolę wartości powrotnej dla odnośnika, dlatego kliknięcie przycisku *Nie* nie spowoduje przejścia do adresu wskazywanego przez odnośnik.

OSTRZEŻENIE

Warto tu zaznaczyć, że okno dialogowe jest tłumaczone przez przeglądarki, dlatego należy unikać stosowania tekstów w rodzaju *Kliknij Tak, aby...*, ponieważ użytkownicy posługujący się systemami w innym języku zobaczą przycisk o innej nazwie.

Prośba o dane użytkownika

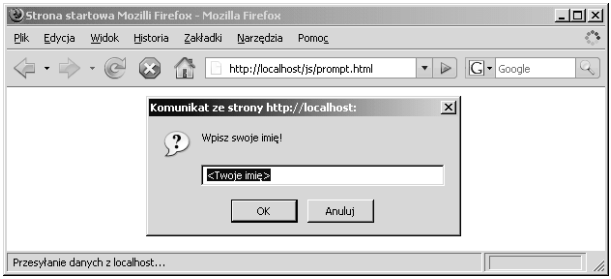
```
var name = window.prompt("Wpisz swoje imię!",
"<Twoje imię>");
```

Prośba o dane użytkownika

Metoda `window.prompt()` pozwala użytkownikowi wpisać tekst w jednowierszowym polu tekstowym (patrz rysunek 2.2). Ta informacja jest wartością zwracaną po wywołaniu tej metody i może zostać wykorzystana w dalszej części skryptu.

```
<script language="JavaScript"
➥type="text/javascript">
var name = window.prompt("Wpisz swoje imię!",
➥"<Twoje imię>");
if (name != null) {
  window.alert("Witaj, " + name + "!");
}
</script>
```

Prośba o wpisanie danych przez użytkownika (prompt.html)



Rysunek 2.2. Okno wprowadzania danych utworzone przez window.prompt()

UWAGA

Po kliknięciu przycisku *Anuluj* lub naciśnięciu klawisza *Escape* `window.prompt()` zwróci wartość `null` (jest to sprawdzane przez powyższy kod). Kliknięcie przycisku *OK* wyświetli wprowadzone dane.