JavaScript. Projekty

Autor: William B. Sanders
Tłumaczenie: Adam Jarczyk
ISBN: 83-7197-811-1
Tytuł oryginału: JavaScript Design
Format: B5, stron: 420

Doskonały przewodnik po tajnikach języka JavaScript, przeznaczony dla tych projektantów WWW, którym przestał wystarczać HTML. Bogato ilustrowana praktycznymi przykładami książka jest kompletnym podręcznikiem najpopularniejszego języka skryptowego, którego znajomość pozwala ożywić strony internetowe. Autor nie zatrzymuje się na czysto wizualnych aspektach zastosowania JavaScriptu. Pokazuje także sposoby komunikowania się JavaScriptu z aplikacjami działającymi po stronie serwera, napisanymi w PHP, ASP czy Perlu, a także odczytywanie i przetwarzanie dokumentów XML.

JavaScript — od podstaw po techniki zaawansowane.

- Składnia JavaScriptu
- Obsługa okienek z ostrzeżeniami i komunikatami
- Efekty podmiany obrazka (rollover) i animacja z wykorzystaniem warstw
- Odczytywanie i wysyłanie ciasteczek (cookies)
- Obsługa ramek i otwieranie nowych okien przeglądarki
- Model dokumentu DOM
- Krótkie wprowadzenie do ASP, Perla i PHP — integracja JavaScriptu ze skryptami działającymi po stronie serwera
- Podstawy VBScript
- Łączenie JavaScriptu z apletami Javy
- Korzystanie z dokumentów XML

„JavaScript projekty", kompendium JavaScriptu, zawierające także wprowadzenie do wielu pokrewnych, przydatnych technologii, to obowiązkowa pozycja na półce twórcy stron internetowych. Ta książka rozszerzy Twoje horyzonty!

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 8.

Programowanie obiektowe w języku JavaScript i model obiektów dokumentu (DOM)

W rozdziale:

- Programowanie obiektowe w języku JavaScript
- Idea prototypu
- Model obiektów dokumentu

Jak dotąd obiekty były omawiane jedynie pod kątem ich budowania i adresowania różnych właściwości i metod. Pod pewnymi względami właściwości zachowują się jak zmienne — możemy przypisywać im wartości, które mogą później być zmieniane. Analogicznie pokazaliśmy, że metody są funkcjami skojarzonymi z obiektami i ich właściwościami. Co jednak ważniejsze, wiele luźnych fragmentów kodu możemy połączyć w obiekt, który zawiera wszystkie informacje udostępniane przez zmienne i czynności właściwe dla funkcji. Jeśli wyobrazimy sobie obiekt jako zbiór *nazwanych* właściwości mających *wartości*, które mogą być *danymi dowolnego typu*, zobaczymy, że są one elementami konstrukcyjnymi dla skryptu. Pamiętając, że funkcje mogą być typem danych, nazywamy je metodami, jeśli są wartością właściwości. Może w tym pomóc myślenie o obiektach jako o *zakodowanych modułach*, które pomagają w organizowaniu skryptów.

JavaScript ma obiekty, a więc musimy pomyśleć o programowaniu z ich użyciem. Ponieważ zakładam, że większość Czytelników tej książki ma raczej doświadczenie w projektowaniu witryn, a nie w programowaniu, to fakt, iż programowanie obiektowe (OOP — *object-oriented programming*) w JavaScripcie różni się od OOP w C++ lub Javie, nie powinien nikogo rozczarować. Zamiast wdawać się w spory, czy JavaScript jest językiem obiektowym, czy nie, zamierzam traktować JavaScript jak *rodzaj języka obiektowego*. Podobnie, jak republikanie i demokraci tworzą odmienne partie polityczne, lecz nadal partie, JavaScript jest typem obiektowego języka programowania, mimo że nie przypomina innych języków OOP. W następnym podrozdziale wszystkie komentarze będą dotyczyły tego, co sprawia, że JavaScript jest językiem obiektowym, i co to oznacza dla osoby piszącej skrypty w sposób obiektowy.

Programowanie obiektowe w języku JavaScript

O obiektach w JavaScripcie możemy myśleć między innymi jak o *tablicach asocjacyjnych (skojarzeniowych)*. Czytelnik mógł się tego spodziewać z przedstawionych wcześniej opisów tablic i z faktu używania instrukcji `for/in` do wydobywania informacji z obiektów. Poniższy skrypt tworzy obiekt, przypisuje do niego dane, a następnie wydobywa *w postaci tablicy*:

objectArray.html

```
<html>
<head>
<title>Tablica skojarzeniowa</title>
<script language="JavaScript">
var tree = new Object();
tree.size = "Duże i wyższe niż wuj Jim.";
tree.shade = "latem w nim jest chłodno.";
tree.fall = "Hodujemy wielkie dynie."
var display = tree["size"] + "<br>";
display += tree["shade"] + "<br>"
display += tree["fall"]
document.write(display);
</script>
</head>
<body bgcolor="wheat" >
</body>
</html>
```

Jak widać, wszystkie właściwości obiektu `tree` są zasadniczo elementami tablicy `tree`. Jest to tablica skojarzeniowa; możemy uznawać obiekty JavaScriptu za obiekty skojarzeniowe.

Używając funkcji, możemy tworzyć obiekty z nich złożone. Takie funkcje noszą nazwę *funkcji konstruktorskich.* Gdy następnie utworzymy obiekt z funkcji konstruktorskiej, używając nowego konstruktora, nowy obiekt będzie miał właściwości w funkcji.

Poniższy przykład wykorzystuje funkcję konstruktorską do utworzenia nowego obiektu o nazwie Bloomfield. Wszystkie właściwości tego obiektu dziedziczą właściwości obiektu Mall.

constructFuncObj.html

```
<html>
<head>
<title>Metoda konstruktorska</title>
<script language="JavaScript">
//Funkcja konstruktorska
function Mall(stName,shop,product) {
      this.stName = stName;
      this.shop = shop;
      this.product = product;
      }
var Bloomfield = new Mall("Bloomy","Nuts and Lightning Bolts","Organic Electricity")
var display = Bloomfield.stName + "<br>";
display += Bloomfield.shop + "<br>";
display += Bloomfield.product;
document.write(display);
</script>
</head>
<body bgcolor="mintcream" >
</body>
</html>
```

Jak zobaczymy na ekranie, wszystkie właściwości obiektu `Mall` zostały odziedziczone przez obiekt `Bloomfield`. Pomocne w zrozumieniu tego może być myślenie o metodach jako o *właściwościach funkcji*, a nie prostych funkcjach dołączonych do obiektu. Dzięki temu lepiej zrozumiemy, że funkcja jest *istotnie* właściwością.

Idea prototypu

Większość różnic pomiędzy JavaScriptem a językami takimi jak Java polega na tym, iż programowanie obiektowe w Javie opiera się na innym typie klasy niż w JavaScripcie. W tradycyjnym programowaniu obiektowym klasa jest zbiorem, natomiast każdy obiekt jest egzemplarzem tej klasy lub zbioru. Jednakże w JavaScripcie pojęcie klasy obraca się dookoła idei *prototypu*. W przeciwieństwie do pojęcia obiektu jako zbioru, gdzie egzemplarz obiektu jest członkiem klasy, idea prototypu traktuje nazwany obiekt biorąc pod uwagę wszystkie właściwości, jakie posiadają wszystkie obiekty przynależące do klasy. Rozważmy na przykład klasę o nazwie Mall.

Obiekt `Mall` (ang. pasaż handlowy) posiada kilka właściwości przynależnych wszystkich pasażom handlowym. Oto ich lista:

- nazwa (Bloomfield Mall),
- sklepy (mięsny, cukiernia, sklep sportowy itd.),
- liczba sklepów (15),

- klienci (liczba klientów lub nazwiska),
- pracownicy (liczba lub nazwiska),
- role (kasjer, kierownik, ochrona, złodziej).

W JavaScripcie pojęcie klasy w zastosowaniu do `Mall` sugeruje, iż `Mall` posiada wszystkie właściwości typowe dla pasażu handlowego. Wobec tego możemy, zajmując się klasą Mall, stosować poniższe właściwości:

```
Mall.name //nazwa
Mall.shop //sklep
Mall.shopNumber //liczba sklepów
Mall.customer //klient
Mall.employee //pracownik
Mall.roles //role
```

Aby zobaczyć różnice pomiędzy zmienną a obiektem, poniższy skrypt definiuje obiekt `Mall` i zmienną `Mall`. Zdefiniowana zmienna została oznaczona jako komentarz za pomocą podwójnego ukośnika (`//`), ponieważ zmienna `Mall` nie będzie działać jako zdefiniowany obiekt:

objChara.html

```
<html>
<head>
<title>Zmienna i obiekt</title>
<script language="JavaScript">
var Mall = new Object();
Mall.shops = "Sklep Vinny'ego - broń i artykuły pierwszej pomocy";
//var Mall;
//Mall.shops = "Salon używanych kijów golfowych i klinika analityczna";
document.write(Mall.shops);
</script>
</head>
<body bgcolor="mintcream" >
</body>
</html>
```

Jeśli usuniemy podwójne ukośniki i zamiast tego oznaczymy w skrypcie jako komentarz oryginalny obiekt `Mall`, zobaczymy, że skrypt nie zadziała, ponieważ zmienna zdefiniowana w drugim przypadku nie utworzyła obiektu. Zamiast tego została utworzona zmienna, a chociaż zmienne mają pewne właściwości (na przykład długość), to nie można przypisywać im właściwości.

Obiekty String i Date

Musimy jeszcze omówić dwa ważne obiekty wbudowane — `String` i `Date`. Każdy z nich ma właściwości, metody i parametry, które trzeba poznać. Obiekt `Date` z punktu widzenia projektu jest mocno wyspecjalizowany, lecz możemy go użyć na wiele ciekawych sposobów, a nie tylko do wyświetlania daty i godziny. Dla projektanta ważniejszy jest obiekt `String` i sposoby, na jakie może posłużyć do formatowania wyświetlanych danych.

Sposób użycia obiektu String

Obiekt `String` posiada mnóstwo metod, lecz na razie omówimy tylko najważniejsze wybrane z nich. Czytelnik może zacząć przyglądać się różnym sposobom formatowania łańcuchów i uczyć się, jak wykorzystać wbudowane właściwości do manipulowania wyglądem strony na ekranie. Na przykład, poniższy skrypt zawiera obiekt łańcuchowy wykorzystujący *równocześnie pięć metod*:

stringMethod.html

```
<html>
<head>
<title>Obiekty i metody łańcuchowe</title>
<script language="JavaScript">
var myWord = new String("Niegrzecznie jest mrugać!");
myWord = myWord.substring(18,22).fontsize(7).italics().fontcolor("red").blink();
document.write(myWord);
//Atrybut blink zrujnował więcej stosunkowo przyzwoitych projektów WWW niż
//jakikolwiek inny i został w końcu wyeliminowany. Do obejrzenia tej
//irytującej metody potrzebna jest starsza przeglądarka (4.0).
</script>
</head>
<body bgColor="peachpuff">
</body>
</html>
```

Proszę zwrócić uwagę, że po zdefiniowaniu obiektu za pomocą konstruktora `String()` możliwe jest zapisanie poniższej instrukcji z pięcioma metodami dołączonymi do samego obiektu łańcuchowego:

```
myWord = myWord.substring(18,22).fontsize(7).italics().fontcolor("red").blink();
```

Na końcu skryptu obiekt łańcuchowy `myWord` posiada właściwości `substring` (podłańcuch), `fontsize` (rozmiar czcionki), `italics` (pochylenie), `fontcolor` (kolor czcionki) — czerwony oraz `blink` — czcionka migająca. Jak widzimy, używając łańcucha jako obiektu możemy kontrolować jego wygląd na stronie.

Proszę nigdy nie używać migających czcionek w prawdziwych projektach! Wywołują one oczopląs — a poza tym działają tylko w Netscape Navigatorze.

Metody łańcuchowe możemy podzielić na trzy kategorie, jak pokazano w tabeli 8.1.

Metody znacznikowe

Metody znacznikowe skojarzone z łańcuchami odpowiadają znacznikom języka HTML. Na przykład metoda `bold()` jest skojarzona ze znacznikiem `<b>` pogrubiającym czcionkę. Analogicznie `fontcolor()` i `fontsize()` są skojarzone ze znacznikiem `<Font>`. Uwaga projektanci — proszę uważać z tymi metodami. Z powodu możliwości i elastyczności kaskadowych arkuszy stylów (CSS) wiele znaczników, takich jak `<Font>`, nie jest już zalecanych. Jeśli przyzwyczaimy się zbytnio do używania metod znacznikowych w JavaScripcie, możemy pewnego dnia zostać daleko w tyle z przestarzałą

Tabela 8.1. *Typy metod łańcuchowych*

Znaczniki	Czynności	Wyrażenia regularne
`Anchor()`	`charAt()`	`match()`
`Big()`	`charCode()`	`replace()`
`Blink()` – tylko w NN	`concat()`	`search()`
`Bold()`	`indexOf()`	`split()`
`Fixed()`	`lastIndexOf()`	
`Fontcolor()`	`slice()`	
`Fontsize()`	`substring()`	
`Italics()`	`substr()`	
`Link()`	`toLowerCase()`	
`Small()`	`toUpperCase()`	
`Strike()`		
`Sub()`		
`Sup()`		

technologią. Proszę przejrzeć rozdział 12., „Dynamiczny HTML", i zobaczyć, co możemy osiągnąć, łącząc CSS z JavaScriptem. Przy okazji zobaczymy, ile wbudowanych metod możemy dodać do łańcucha.

Metody czynnościowe

Używam terminu „metody czynnościowe" do oznaczenia metod przekształcających skład łańcucha lub znajdujących informacje o łańcuchach, lecz nie używających wyrażeń regularnych. Metody te są podstawowymi narzędziami pracy z łańcuchami. Najważniejsze z nich zajmują się identyfikacją podłańcuchów i znaków.

- `substring(początek,koniec)` — podaje pozycję liczbową w łańcuchu pierwszego i ostatniego znaku podłańcucha, który ma zostać wydzielony z łańcucha.
- `charAt(n)` — podaje wartość pozycji liczbowej znaku w łańcuchu.
- `charCodeAt(n)` — podaje wartość pozycji znaku w łańcuchu i zwraca wartość tego znaku (n) w Unicode (ASCII).
- `indexOf(substr)` — podaje fragment (podłańcuch) do znalezienia w łańcuchu i zwraca pozycję pierwszego znaku podłańcucha. `lastIndexOf()` działa tak samo, lecz zaczyna od końca łańcucha.
- `toUpperCase()` i `toLowerCase()` — przekształca łańcuch na same wielkie lub same małe litery.

Poniższy skrypt daje przykład wykorzystania wszystkich powyższych metod, poza indeksowaniem:

stringMethod2.html

```
<html>
<head>
<title>Metoda 2</title>
```

```
<script language="JavaScript">
var quote = new String("Ona wyrażała wszystkie możliwe emocje.");
var dex = quote.toLowerCase().substring(0,2);
var terity = quote.toUpperCase().charAt(22);
dex = dex.concat(terity);
document.write(dex);
</script>
</head>
<body bgColor="lavenderblush">
</body>
</html>
```

Metody wyrażeń regularnych

Ostatnim typem metod łańcuchowych są metody używające wyrażeń regularnych. Jak intensywnie będziemy korzystać z tych metod, to już zależy od konkretnego projektu, lecz wyrażenia regularne są doskonałym narzędziem do pracy z łańcuchami. Proponuję wrócić do opisu wyrażeń regularnych w rozdziale 3., „Dane i zmienne", lub przejrzeć opis CGI i języka Perl w rozdziale 16., „CGI i Perl", gdzie zostały omówione bardziej szczegółowo różne formaty wyrażeń regularnych.

Czterema metodami wyrażeń regularnych dla obiektu łańcuchowego są:

- `match(regexp)` — zwraca podciąg opisany wyrażeniem regularnym lub wartość `null`.
- `replace(regexp,substitute)` —zastępuje wyrażenie regularne łańcuchem `substitute`.
- `search(regexp)` — znajduje pozycję początkową łańcucha opisanego wyrażeniem regularnym.
- `split(regexp)` — ta metoda działa zarówno na łańcuchach, jak i na wyrażeniach regularnych (JavaScript 1.2). Łańcuch jest dzielony w miejscu wyrażenia regularnego (możliwe, że w wielu miejscach), człon z wyrażenia regularnego jest odrzucany, zaś łańcuch jest przekształcany w tablicę, której elementy oddzielane są położeniem odrzuconego podłańcucha.

Poniższy skrypt przedstawia sposób działania wszystkich czterech metod. Proszę zwrócić uwagę, jak metoda `String.search(exp)` została użyta w metodzie `String.slice()` w celu znalezienia składnika z wyrażenia regularnego. Wynik działania skryptu przedstawia rysunek 8.1.

srngRegExpMeth.html

```
<html>
<head>
<style type="text/css">
body {
font-family: verdana;
background-color:#e6c303;
}
h3 {
color:#6dc303;
```

Rysunek 8.1. *Metody wyrażeń regularnych skojarzone z obiektem łańcuchowym są efektywnym dodatkiem do narzędzi formatujących dostępnych w JavaScripcie*

```
background-color:#1a291f;
}
</style>
<title>Metody wyrazen regularnych w lancuchach </title>
<script language="JavaScript">
//string.match()
var matchThis = "Zagraj to jeszcze raz, Sam ";
var stringer1 = matchThis.match(/sam/i);

//string.replace()
var replaceThis = "Ten skrypt jest osadzony w języku Java.";
var stringer2 = replaceThis.replace(/Java/g,"HTML");

//string.search()
var searchThis = "www.sandlight.com";
var stringer3 = searchThis.slice(searchThis.search(/.com/));

//string.split()
var splitThis = "Ona ma dwa poważne problemy."
var stringer4 = splitThis.split(/dwa poważne/i);

var display = "<h3>Metody wyrażeń regularnych w obiektach łańcuchowych</h3>";
display += matchThis + " staje się : " + stringer1 + "<br>"
display += replaceThis + " staje się : " + stringer2 + "<br>"
display += searchThis + " staje się z pomocą string.slice() : " + stringer3
➥+ "<br>"
display += splitThis + " staje się tablicą : " + stringer4[0]+ " " + stringer4[1]
➥+ "<br>"
var showOff = new String(display);
document.write(showOff.fontcolor("#972126"));
</script>
</head>
<body>
</body>
</html>
```

Różne metody wyrażeń regularnych w obiekcie łańcuchowym mogą zaoszczędzić nam nieco czasu na znajdowanie pozycji w łańcuchach. Zamiast szukania podłańcucha lub pozycji za pomocą pętli, warto pomyśleć o zastosowaniu metod wyrażeń regularnych.

Wykorzystanie obiektu Date

Obiekt `Date` posiada 40 metod i 9 argumentów. Jednakże wiele z tych metod może nie być używanych zbyt często (np. dotyczące czasu Greenwich), zaś większość metod pobiera lub ustawia jedno i to samo. Mimo to Czytelnik powinien zapoznać się z metodami i argumentami obiektu `Date`. Tabela 8.2 zawiera ich zestawienie.

Tabela 8.2. *Właściwości obiektu Date*

Argumenty	
`Miliseconds`	Liczba milisekund od 1 stycznia 1970.
`Datestring`	Określona data i czas w formacie łańcuchowym (czas jest opcjonalny).
`Year`	Rok w zapisie czterocyfrowym (np. 2002).
`Month`	Miesiąc numerowany od 0 do 11 (styczeń ma nr 0).
`Day`	Dzień miesiąca numerowany od 1 do 31. (Kto wie, dlaczego dni zaczynają się od 1 a wszystko inne od 0?).
`Hours`	Godzina dnia od 0 do 23.
`Minutes`	Liczba minut od 0 do 59.
`Seconds`	Liczba sekund od 0 do 59.
`Ms`	Liczba milisekund od 0 do 999.
Metody	Większość zwracanych wartości jest definiowanych przez datę w obiekcie `Date`, nawet jeśli nie jest to bieżąca data i czas.
`getDate()`	Zwraca dzień miesiąca.
`getDay()`	Zwraca dzień tygodnia.
`getFullYear()`	Zwraca bieżący rok z obiektu `Date`.
`getHours()`	Zwraca godziny.
`getMiliseconds()`	Zwraca liczbę milisekund od 01.01.1970 do daty określonej w obiekcie.
`getMinutes()`	Zwraca minuty.
`getMonth()`	Zwraca miesiąc w postaci liczby całkowitej.
`getSeconds()`	Zwraca sekundy.
`getTime()`	Zwraca aktualną godzinę w milisekundach.
`getTimezoneOffset()`	Zwraca różnicę w strefach czasowych pomiędzy czasem lokalnym i uniwersalnym (UTC).
`getUTCDate()`	Zwraca dzień miesiąca w UTC.
`getUTCDay()`	Zwraca dzień tygodnia w UTC.
`getUTCFullYear()`	Zwraca bieżący rok w UTC.
`getUTCHours()`	Zwraca godzinę w UTC.
`getUTCMiliseconds()`	Zwraca liczbę milisekund od 01.01.1970 do bieżącej UTC.

Tabela 8.2. *Właściwości obiektu Date (ciąg dalszy)*

Metody	
`getUTCMinutes()`	Zwraca minuty w UTC.
`getUTCMonth()`	Zwraca miesiąc w UTC w postaci liczby całkowitej.
`getUTCSeconds()`	Zwraca sekundy w UTC.
`getYear()`	Zwraca pole roku; metoda nie zalecana (zastąpiona przez `getFullYear()`).
`setDate()`	Ustawia dzień miesiąca.
`setFullYear()`	Ustawia rok.
`setHours()`	Ustawia godzinę.
`setMiliseconds()`	Ustawia milisekundy.
`setMinutes()`	Ustawia minuty.
`setMonth()`	Ustawia miesiąc jako liczbę całkowitą.
`setSeconds()`	Ustawia sekundy.
`setTime()`	Ustawia bieżący czas w milisekundach.
`setUTCDate()`	Ustawia dzień miesiąca w UTC.
`setUTCFullYear()`	Ustawia rok w UTC.
`setUTCHours()`	Ustawia godzinę w UTC.
`setUTCMiliseconds()`	Ustawia pole milisekund w UTC.
`setUTCMinutes()`	Ustawia minuty w UTC.
`setUTCMonth()`	Ustawia miesiąc w UTC jako liczbę całkowitą.
`setUTCSeconds()`	Ustawia sekundy w UTC.
`setYear()`	Ustawia rok; metoda nie zalecana (zastąpiona przez `setFullYear()`).
`toGMTString()`	Konwertuje datę na łańcuch, używając UTC lub GMT.
`toLocaleString()`	Konwertuje datę i czas.
`toUTCString()`	Konwertuje na łańcuch, stosując UTC.
`valueOf()`	Konwertuje na milisekundy.

Obiekt `Date` jest łatwy w użyciu, mimo tak oszałamiającego zbioru opcji. Jednakże w większości przypadków obiekt ten jest stosowany do porównania przeszłej lub obecnej daty z aktualną albo po prostu do wyświetlania daty na ekranie. Za pomocą zegara UTC można tworzyć ciekawe zegary ogólnoświatowe i inne interesujące zdarzenia (events) związane z upływem czasu, więc nie powinniśmy zapominać o metodach dla `Date` podczas umieszczania daty i czasu na stronach WWW. Poniższy skrypt przedstawia przykład, jak obiekt `Date` może posłużyć do przypominania o Walentynkach:

date.html

```
<html>
<head>
<script language="JavaScript">
var NowDate = new Date();
var DateNow = NowDate.getDate();
var MonthNow = NowDate.getMonth();
```

```
//Aby znaleźć Walentynki, potrzebny jest miesiąc 1 i dzień 14.
//Miesiące zaczynają się od 0, więc luty ma numer 1. Dni liczone są
//od 1. I nie zapomnij o prawdziwej kartce walentynkowej!

if (DateNow == 14 && MonthNow == 1) {
      alert("Dzisiaj Walentynki!")
      } else {
      alert ("Zaczekaj do 14 lutego, niepoprawny romantyku!")
}
</script>
</head>
<body bgcolor="pink">
</body>
</html>
```

Po co nam programowanie obiektowe?

To wprowadzenie do programowania obiektowego ma zachęcić do myślenia o elementach JavaScriptu jako obiektach, ich właściwościach i metodach. W większości zastosowań skrypty w JavaScripcie są krótkie; wprawdzie dobra organizacja programowania jest ważna nawet w krótkich skryptach, lecz programowanie obiektowe nie jest w nich niezbędne. Gdy jednak zaczniemy pracować w większych zespołach nad projektami WWW i skrypty zaczną zwiększać objętość, programowanie obiektowe zacznie być coraz ważniejsze. Ponieważ zachęca ono do tworzenia modułowych jednostek programu, jednostki te możemy udostępniać innym pracownikom zespołu i wykorzystywać ponownie, co oznacza, że nie będziemy musieli wyważać otwartych drzwi za każdym razem, gdy siądziemy do pracy nad skryptem.

Dla projektanta witryn WWW idea modułowych elementów projektu, zdatnych do ponownego wykorzystania, jest bardziej oczywista i intuicyjna. Programowanie obiektowe jest podobne. Jeśli potrafimy napisać złożony fragment kodu w postaci modułu, wówczas następnym razem, gdy taki sam kod będzie potrzebny, będziemy mogli albo nieco zmodyfikować kod (np. zmienić wartości argumentów), albo użyć tego samego modułu w innym miejscu. Dzięki temu czas poświęcony na napisanie kodu zwróci się.

Model obiektów dokumentu

W modelu obiektów dokumentu (DOM — *Document Object Model*) JavaScriptu podstawowym dokumentem jest strona HTML-a. Jak już powiedzieliśmy, obiekt `window`, który zawiera ramki, jest na szczycie hierarchii przeglądarki WWW. Jednakże obiekt `Document` zawiera właściwości, z których informacje są wykorzystywane przez JavaScript. Aby odrzeć DOM z tajemniczości, pomyślmy o nim jak o obiekcie `Document` razem z wszystkimi właściwościami, w tym z metodami. Instrukcje, takie jak `document.write()`, składają się z obiektu (`document`) i właściwości lub metody (`write()`), które są częścią modelu. Upraszczając nieco do przesady, możemy powiedzieć, że model obiektów dokumentu JavaScript jest sumą wszystkich właściwości i metod obiektu `Document`, łącznie z tablicami automatycznie generowanymi na stronie HTML, oraz sposobów dostępu do tych obiektów w JavaScripcie.

Właściwości obiektu Document

Patrząc na właściwości obiektu `Document`, możemy odczuć coś w rodzaju *déjà vu* z podrozdziału poświęconego obiektowi `String` z wcześniejszej części tego rozdziału. Zobaczymy pewne podobieństwa, lecz w większości przypadków właściwości (z pominięciem metod) składające się na DOM są unikatowe dla obiektu `Document`. Tabela 8.3 przedstawia właściwości tego obiektu.

Tabela 8.3. *Właściwości obiektu Document*

Nazwa właściwości	Skojarzona właściwość
`alinkColor`	Aktywny kolor wybranego łącza.
`anchors[tablica]`	Każde zaczepienie na stronie HTML jest elementem tablicy.
`applets[tablica]`	Każdy aplet na stronie HTML jest elementem tablicy.
`bgColor`	Kolor tła dokumentu.
`cookie`	Pliki tekstowe, które JavaScript może zapisywać i odczytywać.
`domain`	Właściwość zabezpieczeń, która może być wykorzystana przez dwa lub więcej serwerów WWW, aby zmniejszyć ograniczenia na interakcje pomiędzy stronami WWW.
`embeds[tablica]` (działa również `plugins[tablica]`)	Każdy obiekt osadzony jest elementem tablicy `embeds[]`. Przykładami obiektów osadzonych są rozszerzenia plug-in i pliki *.swf* (patrz rozdział 18., „Flash ActionScript i JavaScript”, w którym omówiono osadzone pliki Flash w JavaScripcie).
`fgColor`	Kolor tekstu.
`forms[tablica]`	Każdy formularz na stronie HTML jest elementem tablicy, a każdy obiekt w formularzu jest elementem elementu `form` (formularz). Rozdział 11., „Formularze”, szczegółowo omawia formularze w JavaScripcie.
`images[tablica]`	Każdy obraz na stronie HTML jest elementem tablicy `images[]` (tablica ta będzie omówiona szczegółowo w dalszej części rozdziału).
`lastModified`	Data ostatniej modyfikacji w formacie łańcuchowym.
`linkColor`	Początkowy kolor łącza przed odwiedzeniem strony (domyślnie niebieski).
`links[tablica]`	Każde łącze jest elementem tablicy, gdy pojawia się w dokumencie.
`location`	Właściwość URL; obecnie podawana jako `URL` (patrz pozycja `URL` w dalszej części tabeli).
`referer`	Poprzednia strona, która zawiera łącze do bieżącej.
`title`	Tytuł dokumentu.
`URL`	Nowa wersja właściwości lokalizacji podająca URL załadowanej strony.
`vlinkColor`	Kolor odwiedzonego łącza.

Poniższy skrypt przedstawia działanie niektórych z właściwości dokumentu. Obiekt `form` jest tablicą, zaś funkcja JavaScriptu adresuje jedyny obiekt tekstowy jako element tablicy formularzy.

bgcolor.html

```
<html>
<head>
<script language="JavaScript">
function message(hue) {
      document.forms[0].elements[5].value=hue;
      }
//Netscape 6+ najwyraźniej zaczyna liczyć elementy od 1 zamiast od zera. W efekcie
//wszystkie odwołania do document.forms[0].elements[5] powodują zaadresowanie
//przycisku. Dziwne.
</script>
<title>Dynamiczne zmiany koloru</title>
</head>
<body>
<H2> Duża czcionka </h2>
<form>
<input type=button value="Czerwony" onClick="document.bgColor='red'"
➥onMouseDown="message('red')">
<input type=button value="Zielony" onClick="document.bgColor='green'"
➥onMouseOver="message('green')">
<input type=button value="Niebieski" onClick="document.bgColor='blue'"
➥onMouseMove="message('blue')"><br>
<input type=button value="Żółta czcionka" onClick="document.fgColor='yellow'"> <br>
<input type=button value="Ceglasta czcionka" onClick="document.fgColor='firebrick'">
<p>
<input type=text>
</form>
</body>
</html>
```

Innymi ważnymi właściwościami obiektu `Document` są obiekty `array` (tablica), `form` (formularz) i `image` (obraz). Formularze będą omówione szczegółowo w rozdziale 11., zaś tablice były opisane w rozdziale 3. Ważnym obiektem tablicy dokumentu, który warto tu przeanalizować dokładniej, jest właściwość `image` (obraz) jako obiekt.

Obiekty obrazów

Obiekt `image` jest jednym z najbardziej interesujących w HTML-u i JavaScripcie. Gdy umieszczamy kolejno obrazy na stronie HTML, korzystając ze znacznika `<img>`, umieszczamy obrazy w tablicy. Nie deklarujemy tablicy, lecz tworzymy ją przez umieszczanie obrazów na ekranie. Tablica ta budowana jest następująco:

```
document.images[0]
document.images[1]
document.images[2]
```

Na stronie HTML te same obrazy wyglądałyby tak:

```
<img src = ”grafikaA.gif”>
<img src = ”grafikaB.gif”>
<img src = ”fotografiaA.gif”>
```

Jedną z właściwości obiektu `image`, jakie możemy zmieniać dynamicznie w JavaScripcie, jest wartość `src`. Następny skrypt wykorzystuje obrazy GIF o rozmiarze jednego piksela przeskalowane do pionowych pasków. Aby utworzyć jednopikselowy GIF, wystarczy otworzyć program graficzny używany do obróbki GIF-ów (np. Photoshop lub Fireworks) i zdefiniować rozmiar obszaru roboczego 1×1 piksel, a następnie powiększyć lupą do rozmiarów pozwalających na pracę z obrazem. Teraz proszę nadać obszarowi roboczemu kolor o jednej z poniższych wartości szesnastkowych:

```
#3a4c4f     #ce6c56     #859eab     #ffeb89
#abc5a8     #8c3227     #debe71
```

Proszę zapisać każdy jednopikselowy GIF pod nazwą od *c1.gif* do *c7.gif*. Poniższy skrypt mówi, gdzie każdy rysunek jest umieszczony i pokazuje, że JavaScript pozwala adresować tablicę obrazów przez adresy w tablicy od 0 do 6. Skrypt adresuje obraz jako część obiektu `Document` w sposób następujący:

```
document.images[0-6].src = nazwaObrazu.src
```

`nazwaObrazu.src` jest częścią obiektu `image` utworzonego w JavaScripcie. Przy tworzeniu obiektu URL (lub nazwa pliku) obrazu źródłowego jest następujący:

```
nazwaObrazu.src = "nazwaRysunku.gif"
```

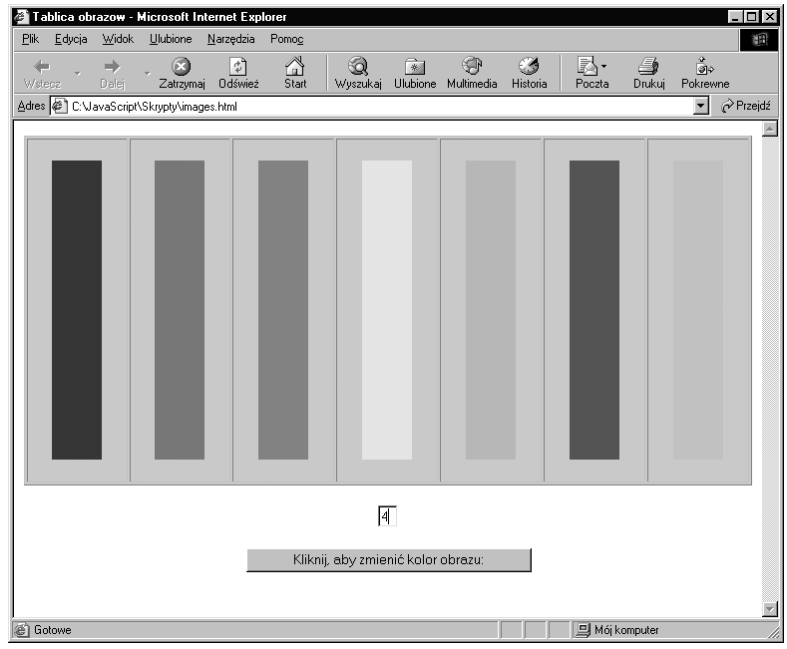
Ponieważ JavaScript potrafi jedynie zmieniać dynamicznie wartość `src` w `document.images[n].src`, nowa wartość musi być zdefiniowana jako `src`, a nie jako sama nazwa obrazu lub URL.

images.html

```
<html>
<head>
<title>Tablica obrazów</title>
<script language="JavaScript">
function changeIt() {
      var c1 = new Image();
      var n = document.forms[0].hue.value;
      n = parseInt(n);
      c1.src = "c3.gif"
      document.images[n].src = c1.src;
}
//Nie wpisywać wartości większych niż 6.
</script>
</head>
<body>
<table border cols=7 width="100%" height="75%" bgcolor="#cccccc" >
<tr>
<td align=center valign=center><img name ="c1" src="c1.gif" border=0 height=300
➥width=50></td>
<td align=center valign=center><img name ="c2" src="c2.gif" border=0 height=300
➥width=50></td>
<td align=center valign=center><img name ="c3" src="c3.gif" border=0 height=300
➥width=50></td>
<td align=center valign=center><img name ="c4" src="c4.gif" border=0 height=300
➥width=50></td>
<td align=center valign=center><img name ="c5" src="c5.gif" border=0 height=300
➥width=50></td>
```

```
<td align=center valign=center><img name ="c6" src="c6.gif" border=0 height=300
➥width=50></td>
<td align=center valign=center><img name ="c7" src="c7.gif" border=0 height=300
➥width=50></td>
</tr>
</table>
<center>
<form>
<input type=text name="hue" size=1><p>
<input type=button value="Kliknij, aby zmienić kolor obrazu:" onClick="changeIt()";>
</form>
</center>
</body>
</html>
```

Rysunek 8.2 przedstawia słupki o różnych kolorach, tak jak będą wyglądać na ekranie, oraz pole do wpisywania wartości elementu tablicy obrazów.



Rysunek 8.2. *Strona HTML zawierająca tablicę obrazów — JavaScript może adresować tablicę i zmieniać wartości src*

Wprawdzie inne właściwości obiektu `Document`, jak np. tablice `links[]` i `anchors[]`, również mogą być adresowane, lecz dla projektanta najważniejsza jest właściwość `images[]`.

Wstępne ładowanie obrazów

Gdy projektujemy stronę, w której jeden obraz jest zastępowany innym, zamiana obrazów musi być natychmiastowa, a od użytkownika nie możemy wymagać cierpliwego oczekiwania na załadowanie nowego obrazu. Na szczęście w JavaScripcie wstępne ładowanie obrazów (do pamięci podręcznej) jest proste. Najpierw definiowany jest nowy obiekt `image`, a następnie definiowane jest źródło nowego obiektu, jak pokazano poniżej:

```
var ciekawyObrazek = new Image();
ciekawyObrazek.src = "nowaFotka.jpg";
```

I to już wszystko. Obraz znajduje się teraz w pamięci podręcznej przeglądarki — został wstępnie załadowany. Pamiętając, że obiekty obrazów mogą być traktowane jak tablica, możemy umieścić wstępnie załadowany obiekt w zdefiniowanym w HTML-u miejscu na obraz. Na przykład, jeśli mamy następujący wiersz w HTML-u:

```
<img src = "firstPix.jpg" name="firstUp">
```

wówczas możemy zastąpić go zbuforowanym obrazem za pomocą poniższego wiersza:

```
document.images[0].src = ciekawyObrazek.src; //images[0] jest pierwszym obrazem
```

lub:

```
document.firstUp.src = ciekawyObrazek.src;
```

Nie istnieje ograniczenie na liczbę obrazów, które możemy załadować do pamięci podręcznej, lecz proszę pamiętać, że pobieranie większej liczby obrazów trwa dłużej. Ponadto podczas procesu ładowania wstępnego możemy zawrzeć również wysokość i szerokość obrazów. Aby dokonać płynnego zastąpienia, obiekty oryginalny i zastępujący powinny mieć te same rozmiary. Na przykład, w poniższy sposób można dobrze dopasować obraz buforowany przez JavaScript z obrazem z HTML-a:

```
<img src = "firstArt" height = 87 width = 55 name ="picasso">
var expArt = new Image(87,55)
```

Wstępnie załadowany obraz będzie miał takie same wymiary jak obraz załadowany przez HTML.

Związek z DOM

W stronie HTML struktura HTML-a zawiera elementy i atrybuty. JavaScript traktuje elementy, np. obrazy i formularze, jak obiekty, a atrybuty elementów jak właściwości. Na przykład, tabela 8.4 przedstawia atrybuty znacznika <img> w HTML-u:

Tabela 8.4. *Atrybuty elementu <img> w HTML-u*

src	alt	longdesc
align	height	width
border	hspace	vspace
usemap	ismap	

W JavaScripcie element <IMG> jest traktowany jak obiekt images[], a wszystkie atrybuty elementu <IMG> jak właściwości obiektu images[]. JavaScript potrafi odczytywać wszystkie właściwości obrazu, lecz może zmienić jedynie właściwość src.

Podobnie jak *wszystkie pozostałe* elementy HTML-a i ich atrybuty pojawiające się na stronie (w dokumencie), stanowią one element składowy modelu obiektów dokumentu (DOM). Stosunek JavaScriptu do obiektów dokumentu jest takie jak elementów HTML-a do właściwości adresowanego obiektu. Gdy więc struktura strony wynika z elementów i atrybutów HTML-a, *zachowanie* strony bierze się ze zdolności JavaScriptu do dynamicznych zmian określonych atrybutów (właściwości) elementów (obiektów).

Podsumowanie

Obiekty są podstawą JavaScriptu, a zrozumienie modelu obiektów dokumentu w JavaScripcie w stosunku do HTML-a jest niezbędne do efektywnego wykorzystania rozlicznych obiektów, jakie możemy znaleźć na stronie WWW. Programowanie obiektowe miało w założeniach pomóc w opanowaniu ogromnych wieloprogramowych projektów, w których trzeba było koordynować tysiące wierszy kodu. W typowych zadaniach, z jakimi spotyka się większość projektantów, potrzeba programowania obiektowego nie polega na konieczności zapanowania nad gigantycznym projektem, ponieważ większość JavaScriptu zajmuje się prostymi zadaniami wykonywanymi na pojedynczych stronach. Jednakże nawet drobne postępy w kierunku programowania obiektowego pomogą lepiej zrozumieć zjawiska zachodzące na naszej stronie WWW i łatwiej osiągać założone efekty.

Projektant stron WWW powinien być w stanie wyobrazić sobie wyniki działania dynamicznej strony WWW, a następnie ożywić stronę za pomocą JavaScriptu. Poznając założenia programowania obiektowego oraz pojęcia obiektów, właściwości i metod w JavaScripcie, zrobiliśmy olbrzymi krok w tym kierunku. Proszę nie uważać programowania obiektowego za ciężar, lecz raczej za okazję do lepszego zrozumienia produktów własnej wyobraźni. Analogicznie, proszę nie traktować programowania obiektowego jako ograniczeń własnej kreatywności. Czasami Czytelnik nie będzie w stanie znaleźć rozwiązania problemu z użyciem programowania obiektowego, lecz do rozwiązania możemy dojść, wykorzystując zmienne i instrukcje nie zawierające obiektów łączonych z właściwościami i metodami. Za to nie grozi kara więzienia! Robiąc drobne kroczki w stronę programowania obiektowego, w końcu nauczymy się wykorzystywać w pełni jego możliwości.