Błyskawicznie opanuj
nowy język programowania!

JavaScript

dla programistów PHP

HELION

O'REILLY®

Stoyan Stefanov

Tytuł oryginału: JavaScript for PHP Developers

Tłumaczenie: Rafał Jońca

ISBN: 978-83-246-8217-1

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/javphp
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 6.

ECMAScript 5

Do tej pory książka opisywała tak naprawdę standard ECMAScript 3, ponieważ to najbardziej rozpowszechniona wersja języka. Najnowszym standardem jest jednak ECMAScript 5.1 (wersja 4. okazała się ślepą uliczką i została zarzucona). Większość nowoczesnych przeglądarek obsługuje ES5. Warto wiedzieć, co przyniosła nowa wersja standardu, nawet jeśli cały czas trzeba pisać kod działający również w starszych przeglądarkach.

Trzema głównymi obszarami aktualizacji są:

- wprowadzenie trybu ścisłego,
- zmiany w tworzeniu obiektów i atrybuty właściwości,
- wprowadzenie nowych API.

Tryb ścisły

Tryb ścisły nie jest zgodny wstecz, więc trzeba jawnie wskazać chęć jego użycia. Raz na funkcję lub też raz na cały program (plik) można wskazać chęć przejścia w tryb ścisły, używając poniższego wiersza:

```
"use strict";
```

Ponieważ to zwykła instrukcja zawierająca tekst, starsze przeglądarki zignorują taki wpis i nie zgłoszą błędu. Interpreter obsługujący tryb ścisły zobaczy wpis z prośbą o użycie trybu ścisłego, więc wyłączy niektóre funkcje i konstrukcje języka JavaScript, które przez ostatnie lata okazały się przynosić więcej złego niż pożytku. W niniejszej książce pominąłem wszystkie niezalecane praktyki, więc nie musisz unikać żadnej z prezentowanych tu konstrukcji.

Oto przykłady funkcjonalności, które w trybie ścisłym spowodują zgłoszenie błędu:

- użycie instrukcji `with`,
- użycie niezadeklarowanych zmiennych,
- użycie `arguments.callee` lub `arguments.caller`,
- próba przypisania czegokolwiek do właściwości tylko do odczytu (np. `window.Infinity = 0;`),
- próba usunięcia niekonfigurowalnych właściwości (wkrótce wyjaśnię, co oznacza **konfigurowalność** właściwości),
- literały obiektów ze zduplikowanymi właściwościami,
- parametry funkcji ze zduplikowanymi nazwami (ponieważ, co zadziwiające, w ES3 zapis `function (a, a, a) {}` jest uznawany za prawidłowy).

Poniżej znajduje się krótki przykład ilustrujący różnicę w działaniu między trybem ścisłym i zwykłym:

```
// Tryb zwykły.
var obj = (function () {
   return {a: 1, a: 2};
})();
obj; // {a: 2}

// Tryb ścisły.
var obj = (function () {
   "use strict";
   return {a: 1, a: 2};
})(); // SyntaxError: zduplikowana właściwość.
```

Atrybuty właściwości

Drugim obszarem zmian w ES5 jest wprowadzenie atrybutów właściwości. Tak naprawdę atrybuty te istniały w języku od dawna, ale nie były bezpośrednio dostępne dla programisty (nie można ich było odczytać lub zmienić).

Właściwość ma atrybut `value` (zawierający wartość) oraz trzy inne atrybuty typu logicznego, które określają, czy właściwość jest:

- wyliczalna,
- zapisywalna,
- konfigurowalna.

Atrybuty właściwości definiuje się przy użyciu specjalnych obiektów nazywanych **deskryptorami**. Oto przykład:

```
var stealth_descriptor = {
   value: "nie mogę jej dotknąć",
   enumerable: false,  // Nie pojawi się w pętlach for-in.
   writable: false,    // Nie można zmienić jej wartości.
   configurable: false // Nie można jej usunąć lub zmienić jej atrybutów.
};
```

Właściwości z wyłączoną konfiguracyjnością mogą mieć atrybut `writable` ustawiony tylko na wartość `false`.

Deskryptory właściwości zapewniają dodatkowy poziom kontroli nad zmianami dotyczącymi właściwości. Umożliwiają między innymi utworzenie stałych, których nie można usunąć. Choć wiele elementów nowego API ES5 można z powodzeniem zaimplementować w ES3, deskryptorów właściwości nie można emulować. W ES3 wszystkie tworzone właściwości można dowolnie modyfikować i usuwać.

Dalsze informacja, a także dodatkowe przykłady dotyczące deskryptorów pojawią się w dalszej części rozdziału.

Nowe API dotyczące obiektów

Nowe API obiektów dotyczy przede wszystkim obsługi atrybutów właściwości i dodatkowo definiuje kilka nowych metod:

- `Object.create()`,
- `Object.defineProperty()`,
- `Object.defineProperties()`,
- `Object.getOwnPropertyDescriptor()`.

Metoda Object.create()

Metoda służy do utworzenia nowego obiektu, ale jednocześnie potrafi zdefiniować dla niego:

- dziedziczenie,
- właściwości obiektu,
- atrybuty właściwości.

Rozważmy następujący fragment kodu:

```
var human = {name: "Jan"};
var programmer = Object.create(
   human,
   {
      secret: stealth_descriptor,
      skill: {value: "Prawdziwy ninja kodu"}
   }
);
```

Obiekt `programmer` dziedziczy po obiekcie `human` przy użyciu `__proto__`, więc nie zawiera właściwości `name` sam w sobie, ale otrzymuje ją dzięki łańcuchowi prototypów:

```
programmer.name; // "Jan"
programmer.hasOwnProperty('name'); // false
programmer.__proto__.hasOwnProperty('name'); // true
```

Dwie pozostałe właściwości są jego własnymi właściwościami:

```
programmer.hasOwnProperty('secret'); // true
programmer.hasOwnProperty('skill');  // true
```

W przypadku właściwości `secret` zastosowano pełny deskryptor umieszczony w zmiennej `stealth_descriptor`. Ustawieniu uległa wartość oraz trzy atrybuty.

Właściwość `skill` definiuje jedynie wartość. Oznacza to, że wszystkie atrybuty zostaną ustawione na swoje wartości domyślne, czyli `false`.

Ustawienie właściwości bez użycia deskryptora spowoduje, że wszystkie atrybuty przyjmą wartość `true`, podobnie jak w ES3:

```
programmer.likes = ['pizza', 'piwo', 'kawa'];
```

Metoda `Object.create()` przypomina użycie funkcji `begetObject()` (opisywanej w rozdziale 4. w podrozdziale „Dziedziczenie”) wraz z dodaną obsługą deskryptorów.

Wcześniej w książce wskazałem, że „puste” obiekty (na przykład `var o = {};`) nie są tak naprawdę puste, ponieważ dziedziczą po `Object.prototype` metody takie jak `toString()`. W ES5 można utworzyć naprawdę pusty obiekt, używając poniższego kodu:

```
var o = Object.create(null); // Nie dziedzicz po niczym.
typeof o.toString;           // "undefined"
```

Metoda Object.getOwnPropertyDescriptor()

Metoda `Object.getOwnPropertyDescriptor()` umożliwia sprawdzenie zawartości deskryptorów:

```
Object.getOwnPropertyDescriptor(programmer, 'secret').configurable; // false
Object.getOwnPropertyDescriptor(programmer, 'likes').configurable;  // true
```

Metody Object.defineProperty() i Object.defineProperties()

Metody `Object.defineProperty()` i `Object.defineProperties()` umożliwiają zdefiniowanie właściwości wraz z deskryptorem już po utworzeniu obiektu:

```
Object.defineProperty(programmer, 'hello', stealth_descriptor);

Object.defineProperties(programmer, {
   goodbye: stealth_descriptor,
   bye: stealth_descriptor
});
```

Ograniczenie zmian obiektów

W ES3 można bez przeszkód zmienić wszystkie obiekty (poza kilkoma wyjątkami dotyczącymi obiektów wbudowanych). Nie zawsze jest to dobre rozwiązanie, ponieważ użytkownicy obiektów mogą je zmienić do tego stopnia, że przestaną one prawidłowo funkcjonować. W ES5 możliwe jest ograniczenie dostępu do obiektów. Niektóre właściwości można przełączyć w tryb tylko do odczytu, ustawiając atrybut `writable` na wartość `false`.

Możliwe jest również ograniczenie dostępu do obiektu jako całości:

- wyłączenie możliwości jego rozszerzania (czyli nie będzie możliwe dodanie nowych właściwości);
- „zapieczętowanie" obiektu, czyli poza wyłączeniem możliwości rozszerzania również zamiana wszystkich istniejących właściwości na niekonfigurowalne (więc nie będzie się ich dało usunąć);
- „zamrożenie" obiektu, czyli „zapieczętowanie" połączone z dodatkowym ustawieniem wszystkich właściwości jako tylko do odczytu.

Rozważmy następujący standardowy obiekt:

```
var pizza = {
   tomatoes: true,
   cheese: true
};
```

Zawsze można do niego dodać nowe właściwości:

```
pizza.pepperoni = 'dużo';
```

Można zmieniać lub usuwać właściwości, ponieważ wszystkie są konfigurowalne i zapisywalne:

```
pizza.cheese = 'mozzarella';
delete pizza.pepperoni; // true
```

Po wyłączeniu możliwości rozszerzania nie uda się dodać nowych właściwości. Metoda `Object.isExtensible()` zwraca wartość `true`, jeśli obiekt można rozszerzać. Metoda `Object.preventExtensions()` zabezpiecza obiekt przed dalszym rozszerzaniem:

```
Object.isExtensible(pizza);       // true
Object.preventExtensions(pizza); // Zwraca obiekt 'pizza'.
Object.isExtensible(pizza);       // false

pizza.broccoli = 'och, nie'; // Błąd, nie można dodać nowych właściwości.
typeof pizza.broccoli;       // "undefined"
```

Wyłączenie usuwania umożliwia metoda `seal()`:

```
Object.isSealed(pizza); // false
Object.seal(pizza);     // Zwraca obiekt 'pizza'.
Object.isSealed(pizza); // true

delete pizza.cheese; // Błąd, nie można usunąć.
pizza.cheese;        // "mozzarella"
```

Nadal jednak możliwa jest zmiana wartości istniejących właściwości:

```
pizza.cheese = 'ricotta';
```

Po wywołaniu metody `freeze()` wszystkie właściwości staną się niezapisywalne:

```
Object.isFrozen(pizza); // false
Object.freeze(pizza);   // Zwraca obiekt 'pizza'.
Object.isFrozen(pizza); // true

pizza.cheese = "gorgonzola"; // Błąd, 'cheese' jest właściwością tylko do odczytu.
pizza.cheese;  // "ricotta"
```

Podsumujmy:

- Metoda `freeze()` wykonuje te same zadania co metoda `seal()` oraz dodatkowo ustawia atrybut `writable` wszystkich właściwości na wartość `false`.
- Metoda `seal()` wykonuje te same zadania co `preventExtensions()` oraz dodatkowo ustawia atrybut `configurable` wszystkich właściwości na wartość `false`.

- Metoda `preventExtensions()` nie ustawia żadnych atrybutów, ale po jej wywołaniu do obiektu nie można dodać żadnych nowych właściwości.
- Żadnej z powyższych operacji nie można cofnąć (innymi słowy, nie ma metod `unfreeze()`, `defrost()` lub `allowExtensions()`).

Pętle alternatywne

W ES3, by otrzymać listę wszystkich właściwości (czyli wynik podobny do funkcji `array_keys()` z PHP), trzeba było skorzystać z pętli `for-in`. W ES5 dostępne są dwie dodatkowe metody: `Object.keys()` i `Object.getOwnProperty➥Names()`. Metoda `keys()` zwraca wszystkie wyliczane właściwości (czyli takie, które pojawiłyby się w pętli `for-in`), natomiast `getOwnPropertyNames()` zwraca wszystkie własne właściwości niezależnie od statusu ich wyliczania.

Obie metody działają w zasadzie identycznie, jeśli nie używa się deskryptorów do zmiany atrybutów `enumerable`:

```
var pizza = {tomatoes: true, cheese: true};
Object.keys(pizza);                 // ["tomatoes", "cheese"]
Object.getOwnPropertyNames(pizza); // ["tomatoes", "cheese"]
```

Obie metody zwracają tylko właściwości własne, więc nie biorą pod uwagę właściwości dziedziczonych po prototypie. Utwórzmy obiekt `pizza_v20`, który dziedziczy po `pizza`, i dodajmy do niego nowe właściwości:

```
var pizza_v20 = Object.create(pizza, {
   salami: {value: "lots", enumerable: true},
   sauce: {value: "secret"}
});
```

Metoda `keys()` nie zwróci `sauce`, ponieważ nie jest to właściwość wyliczana, ale metoda `getOwnPropertyNames()` zwróci wszystkie właściwości:

```
Object.keys(pizza_v20);                 // ["salami"]
Object.getOwnPropertyNames(pizza_v20); // ["sauce", "salami"]
```

Zauważ, że wyniki nie zawierają właściwości obiektu `pizza`, bo stanowią element dziedziczenia:

```
pizza_v20.cheese;              // true
pizza_v20.__proto__ === pizza; // true
```

Metoda Object.getPrototypeOf()

Metoda `Object.getPrototypeOf()` to już ostatnia z nowości konstruktora `Object()` w ES5. W ES3 inspekcja możliwa była tylko przy użyciu metody `isPrototypeOf()`,

więc trzeba było zgadnąć obiekt nadrzędny, by sprawdzić, czy to faktycznie on jest prototypem.

W ES5 można zapytać bezpośrednio: „Kto jest twoim prototypem?”:

```
Object.getPrototypeOf(pizza_v20) === pizza; // true
```

W zasadzie metoda ta zwraca taki sam wynik jak właściwość `__proto__`, ale warto pamiętać, że `__proto__` nie jest standardem (i nie jest dostępne w przeglądarce IE). ES5 przyznaje, że istnieje potrzeba właściwości `__proto__` w wersji przynajmniej do odczytu, więc pojawiła się metoda `getPrototypeOf()`. Wersja z możliwością zapisu nadal jest źródłem gorących debat.

Nowe API dotyczące tablic

ES5 wprowadza również kilka przydatnych metod do `Array.prototype`, a także jedną metodę do konstruktora `Array`.

Metoda Array.isArray()

Metoda `Array.isArray()` to wygodny (bez uciekania się do dodatkowych sztuczek) sposób na stwierdzenie, czy obiekt jest tablicą, czy zwykłym obiektem (pamiętaj, że tablice **są** obiektami).

Jeśli chcesz utworzyć tak zwany **shim** (uzupełnienie) metody i korzystać z niego również w ES3, skorzystaj ze sztuczki `Object.prototype.toString()` w sposób pokazany we wcześniejszej części książki:

```
if (!Array.isArray) {
   Array.isArray = function (candidate) {
      return Object.prototype.toString.call(candidate) === '[object Array]';
   };
}

Array.isArray([]); // true
```

Metoda jest równoważna funkcji `is_array()` z PHP.

Metody indexOf() i lastIndexOf()

Dwie nowe metody `Array.prototype.indexOf()` i `Array.prototype.lastIndexOf()` oferują nowe możliwości związane z przeszukiwaniem zawartości tablic:

Metoda `Array.prototype.indexOf()` zwraca indeks pierwszego wystąpienia elementu:

```
var a = ['raz', 'i', 'dwa', 'i', 'trzy', 'i', 'cztery'];
a.indexOf('trzy'); // 4
```

Przypomina w działaniu funkcję `array_search()` z PHP.

Gdy metoda `Array.prototype.indexOf()` zwraca pierwsze wystąpienie, metoda `Array.prototype.lastIndexOf()` zwraca ostatnie:

```
var a = ['raz', 'i', 'dwa', 'i', 'trzy', 'i', 'cztery'];
a.indexOf("i");     // 1
a.lastIndexOf("i"); // 5
```

Wyszukiwanie korzysta ze ścisłego porównywania wartości (czyli uwzględnia również typ wartości):

```
[1, 2, 100, "100"].indexOf("100"); // 3
[1, 2, 100, "100"].indexOf(100);   // 2
```

Metoda `indexOf()` przyjmuje dodatkowo indeks pozycji startowej, a metoda `lastIndexOf()` indeks pozycji końcowej:

```
var arr = [100, 1, 2, 100];

arr.indexOf(100);     // 0
arr.indexOf(100, 2);  // 3

arr.lastIndexOf(100);    // 3
arr.lastIndexOf(100, 2); // 0
```

Przechodzenie przez elementy tablicy

W ES5 pojawiła się metoda `Array.prototype.forEach()`, która umożliwia przejście w pętli przez wszystkie elementy tablicy bez potrzeby stosowania instrukcji pętli. Przyjmuje funkcję wywołania zwrotnego, która potrafi wykonać konkretną operację na elemencie tablicy lub też całej tablicy.

Oto bardzo prosty przykład, wyświetlający wszystkie argumenty przekazane do funkcji wywołania zwrotnego:

```
["a", "b", "c"].forEach(function () {
   console.log(arguments);
});
```

W konsoli pojawi się następujący wynik:

```
["a", 0, Array[3]]
["b", 1, Array[3]]
["c", 2, Array[3]]
```

Jak łatwo się domyślić, argumentami są:

```
[aktualny element, jego indeks, cała tablica]
```

Filtrowanie

ES5 wprowadza metodę `Array.prototype.filter()`, która przypomina w działaniu funkcję `array_filter` z PHP i umożliwia wykonanie funkcji zwrotnej dla każdego elementu tablicy. Jeśli wywołanie zwrotne zwróci wartość `true`, element trafia do nowej tablicy, która zostanie zwrócona po sprawdzeniu wszystkich elementów:

```
function testVowels(char) {
   return (/[aeiou]/i).test(char);
}

var input = ["a", "b", "c", "d", "e"];
var output = input.filter(testVowels);

output.join(', '); // "a, e"
```

Sprawdzanie zawartości tablicy

Dwie nowe metody, `every()` i `some()`, umożliwiają analizę zawartości tablicy i zwrócenie wartości logicznej, jeśli elementy tablicy spełniły test określony w wywołaniu zwrotnym lub go nie spełniły.

Przypuśćmy, że chcemy sprawdzić, czy tablica zawiera liczby parzyste. Funkcja testu ma postać:

```
function isEven(num) {
   return num % 2 === 0;
}
```

A sam test wygląda następująco:

```
// Czy *wszystkie* liczby są parzyste?
[1, 2, 4].every(isEven); // false

// Czy niektóre (przynajmniej jedna) liczby są parzyste?
[1, 2, 4].some(isEven); // true
```

Odwzorowanie i redukcja

W ES5 pojawiły się metody przypominające w działaniu funkcje `array_map()` i `array_reduce()` z PHP: `Array.prototype.map()` i `Array.prototype.reduce()`, a także metoda dodatkowa `Array.prototype.reduceRight()`.

Metoda `map()` zwraca nową tablicę, której elementy zostały zmienione przez przekazaną funkcję wywołania zwrotnego.

Oto funkcja, która zmienia znaki na ich odpowiedniki w postaci encji HTML:

```
function entity(char) {
   return "&#" + char.charCodeAt(0) + ";";
}
```

Zastosujmy ją dla wszystkich elementów tablicy:

```
var input = ['a', 'b', 'c'];
var out = input.map(entity);
out; // ["&#97;", "&#98;", "&#99;"]
```

Metoda `Array.prototype.reduce()` przyjmuje tablicę, a następnie konwertuje ją do pojedynczej wartości, używając funkcji wywołania zwrotnego.

Przypuśćmy, że chcemy zsumować wszystkie wartości w tablicy i dodać 100 do wyniku. Zaczynamy więc od 100 i w każdym kolejnym kroku do sumy dodajemy nową wartość:

```
function sum(running_sum, value, index, array) {
   console.log(arguments);
   return running_sum + value;
}

[1, 2, 3].reduce(sum, 100); // 106
```

Po wykonaniu kodu w konsoli pojawią się następujące wpisy:

```
[100, 1, 0, Array[3]]
[101, 2, 1, Array[3]]
[103, 3, 2, Array[3]]
```

Metoda `Array.prototype.reduceRight()` działa w ten sam sposób, ale analizuje elementy tablicy od prawej do lewej, czyli zaczyna od ostatniego elementu i przesuwa się w stronę pierwszego.

W tym przykładzie wynik końcowy będzie identyczny, ale w konsoli pojawią się inne wartości:

```
[1, 2, 3].reduceRight(sum, 100); // 106
```

W konsoli pojawią się wpisy:

```
[100, 3, 2, Array[3]]
[103, 2, 1, Array[3]]
[105, 1, 0, Array[3]]
```

Przycinanie tekstu

ES5 wprowadziło metodę `String.prototype.trim()`, przypominającą w działaniu metodę `trim()` z PHP:

```
" witaj ".trim(); // "witaj"
```

To jedyna metoda dodana do obsługi tekstów w ES5.

Niektóre środowiska oferują również metody `trimLeft()` i `trim➥Right()`, ale nie stanowią one części standardu.

Nowości w obiekcie Date

Do obsługi dat w języku JavaScript dodano trzy nowe metody.

Metoda `Date.now()` to równoważnik `(new Date()).getTime()` lub `+new Date()`. Zwraca aktualny czas w postaci znacznika czasowego:

```
Date.now(); // 1327943271496
Date.now() === +new Date(); // true
```

Metoda `Date.prototype.toISOString()` to kolejna z metod odpowiedzialnych za konwersję daty do formatu tekstowego:

```
var date = new Date(2012, 11, 31);
date.toDateString(); // "Mon Dec 31 2012"
date.toISOString();  // "2012-12-31T08:00:00.000Z"
```

Metoda `toISOString()` przydaje się do reprezentacji dat w formacie JSON. Używa jej metoda `JSON.stringify()`, o której za chwilę. W zasadzie istnieje również metoda `Date.prototype.toJSON()`, która zwraca dokładnie taki sam wynik jak metoda `toISOString()`.

```
var today = new Date();
today.toJSON() === today.toISOString(); // true
```

Metoda Function.prototype.bind()

Ponieważ funkcje są obiektami, które można dowolnie przekazywać, a `this` wewnątrz funkcji zależy od sposobu jej wywołania, czasem ryzykowne jest użycie `this`, ponieważ nie ma się pewności, co tak naprawdę zawiera. Rozważmy następujący przykład:

```
var breakfast = {
   drink: "kawa",
   eat: "bekon",
   my: function () {
      return this.drink + " i " + this.eat;
   }
};
breakfast.my(); // "kawa i bekon"
```

Jak na razie żadnych niespodzianek. Przypuśćmy jednak, że utworzymy referencję do metody i ją wywołamy:

```
var morning = breakfast.my;
morning(); // "undefined i undefined"
```

W tym przypadku `this` to obiekt globalny, który nie zawiera właściwości `drink` lub `eat`. Można jednak powiązać metodę `my()` z obiektem `breakfast`, by wszystko działało zgodnie z oczekiwaniami:

```
var morning = breakfast.my.bind(breakfast);
morning(); // "kawa i bekon"
```

Obiekt JSON

Kończąc dyskusję na temat ECMAScript 5, warto wspomnieć o obiekcie `JSON`. To dodatek do ES5 standaryzujący coś, co zostało zaimplementowane we wszystkich przeglądarkach internetowych, włączając IE8 lub nowszą. JSON to skrót od *JavaScript Object Notation* (notacja obiektowa JavaScript) i stanowi standard wymiany danych używający literałów obiektów i tablic JavaScript do kodowania dowolnych danych.

Obiekt JSON posiada dwie (niezbyt szczęśliwie nazwane) metody — `stringify()` i `parse()` — które odpowiadają działaniu funkcji `json_encode()` i `json_decode()` z PHP:

```
JSON.stringify({hello: "world"});       // '{"hello":"world"}'
JSON.parse('{"hello":"world"}').hello; // "world"
```

W środowisku, które nie posiada wbudowanej obsługi JSON, można użyć biblioteki dostępnej pod adresem *https://github.com/douglascrockford/JSON-js*.

Shimy

„Shim" lub „polyfill" to nazwa stosowana w przypadku udostępniania nowego API w starszym środowisku. Ponieważ JavaScript umożliwia zmianę wbudowanych obiektów i ich prototypów, emulacja nowego API jest bardzo prosta w realizacji. Oto przykład shimu:

```
if (!Date.now) {
   Date.now = function () {
      return new Date().getTime();
   };
}
```

I jeszcze jeden przykład:

```
// źródło:
// https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/String/Trim
if (!String.prototype.trim) {
   String.prototype.trim = function () {
      return this.replace(/^\s+|\s+$/g, '');
   };
}
```

Shimy są najczęściej bardzo krótkie, więc można je wczytać przed wykonaniem pierwszego wiersza właściwego kodu. Dzięki temu w starych przeglądarkach łatwo uzyskać nowe funkcjonalności.

Warto jednak zdawać sobie sprawę z tego, że nie wszystkie funkcjonalności można zrealizować w ten sposób. Przykładem są tryb ścisły i wszystkie metody dotyczące deskryptorów właściwości.

Aby sprawdzić, które funkcje ES5 są obsługiwane w poszczególnych przeglądarkach, skorzystaj ze strony *http://kangax.github.io/es5-compat-table/*.

Skrowidz

E

F

G

H

I

J

K

L, Ł

M

N

O

P

R

S, Ś

T

U

W

X

Z

Programowanie

O'REILLY®

JavaScript dla programistów PHP

Czasy wąskiej specjalizacji programistów odchodzą w przeszłość. Współczesne projekty wymagają od nich szerokiej wiedzy, dotyczącej zarówno języków działających po stronie serwera (Java, PHP), jak i przeglądarki (JavaScript). Jest jednak światełko w tunelu — jeśli znasz tylko jeden z języków programowania, jesteś w stanie sprawnie opanować kolejny. Dzięki tej książce to zadanie stanie się prostsze!

Książka jest przeznaczona dla programistów PHP chcących szybko nauczyć się programowania w języku JavaScript. W trakcie lektury poznasz różnice i podobieństwa między tymi dwoma językami. W kolejnych rozdziałach rozgryziesz składnię JavaScriptu i błyskawicznie zaczniesz używać funkcji wbudowanych w ten język. Ponadto opanujesz programowanie obiektowe oraz sposoby jego wykorzystania. Na koniec nauczysz się testować kod napisany w JavaScripcie. To najlepszy podręcznik do nauki tego języka dla osób znających podstawy PHP.

Dzięki tej książce:

- poznasz składnię języka JavaScript
- będziesz programować w nim obiektowo
- zaznajomisz się z funkcjami wbudowanymi
- błyskawicznie poznasz kolejny język programowania

Poznaj możliwości języka JavaScript i wykorzystaj je w Twoim projekcie!

ISBN 978-83-246-8217-1

Cena 32,90 zł

Informatyka w najlepszym wydaniu

9 788324 682171