PROGRAMUJ W JAVASCRIPCIE JAK PROFESJONALISTA!

Apress

Zaawansowane techniki języka JavaScript

Wydanie II

John Resig, Russ Ferguson, John Paxton

Helion

Tytuł oryginału: Pro JavaScript Techniques, Second Edition

Tłumaczenie: Rafał Szpoton

ISBN: 978-83-283-2086-4

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/ztejs2.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/ztejs2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 1

Profesjonalne techniki w JavaScript

Zapraszamy do lektury książki pt. *Zaawansowane techniki języka JavaScript. Wydanie II*. Zawiera ona przegląd obecnego stanu technologii języka JavaScript do wykorzystania przez zawodowych programistów. A kim jest zawodowy programista? Jest nim ktoś, kto dobrze rozumie podstawy języka JavaScript (a także prawdopodobnie kilku innych języków) oraz jest zainteresowany jego wszechstronnością. Chciałby poznać dokładniej Obiektowy Model Dokumentu DOM (ang. *Document Object Model*), a także zapoznać się z tym całym zamieszaniem wokół dyskusji na temat wzorca Model-Widok-Kontroler (ang. *Model-View-Controller*, MVC) po stronie klienta. W książce tej znajdzie również uaktualnione informacje na temat interfejsów API, nowe funkcjonalności, a także kreatywne zastosowania zaprezentowanego kodu.

To już drugie wydanie tej książki. Od czasu jej pierwszego wydania w 2006 r. wiele rzeczy uległo zmianie. W tamtym czasie język JavaScript przechodził nieco bolesną metamorfozę od nieprofesjonalnego języka skryptowego używanego dla zabawy do postaci wydajnej i przydatnej do wielu różnych zadań. W sumie możesz ten etap uznać za wiek dojrzewania tego języka. Obecnie język JavaScript znajduje się podczas kolejnej transformacji, a używając naszej metafory, przechodzi od wieku dojrzewania do dojrzałości. Zastosowanie języka JavaScript jest niemalże powszechne. Na swojej głównej stronie używa go (w zależności od statystyk) od 85 do 95 procent stron internetowych. Wielu ludzi uznaje język JavaScript za najbardziej popularny na świecie język programowania (biorąc pod uwagę liczbę osób, które używają go regularnie). Jednak zdecydowanie bardziej istotne od wskaźników użycia są jego efektywność oraz możliwości.

JavaScript przebył drogę od języka służącego do celów zabawowych (przewijanie obrazków, modyfikacja tekstu na pasku stanu), poprzez etap efektywnego, jednak ograniczonego narzędzia (przypomnij sobie walidację formularzy po stronie klienta), aż do swojej obecnej postaci jako powszechnie wykorzystywanego języka programowania nieograniczonego już dłużej ciasnymi granicami. Używając JavaScriptu, programiści tworzą narzędzia udostępniające funkcjonalności wzorca MVC, co przez długi czas było domeną oprogramowania po stronie serwera. Tworzą również złożone wizualizacje danych, biblioteki szablonów itp. Lista zastosowań zwiększa się cały czas. Aby udostępnić w pełni funkcjonalne bogate interfejsy, które korzystają z danych po stronie serwera, programiści musieli dawniej polegać na języku .Net lub też klientach napisanych w Java Swing, a obecnie można to samo osiągnąć, korzystając z języka JavaScript w przeglądarce. Używając Node.js, możemy dodatkowo uzyskać własną wersję maszyny wirtualnej działającej w JavaScript, w której można uruchomić wiele różnych aplikacji napisanych w tym języku i żadna z nich nie będzie wymagać do działania przeglądarki.

W tym rozdziale opiszemy zarówno drogę, która doprowadziła nas do obecnego etapu w rozwoju tego języka, jak i to, dokąd on zmierza. Poznamy wiele różnych usprawnień wprowadzonych do technologii przeglądarek (a także przyczyny ich popularności), które mają wpływ na rewolucję w języku JavaScript.

Konieczne będzie również przyjrzenie się samemu stanowi tego języka, ponieważ zanim spojrzymy w przyszłość, chcielibyśmy wiedzieć, gdzie znajdujemy się obecnie. W kolejnych rozdziałach dowiemy się, czego potrzebuje zawodowy programista JavaScript, aby móc w pełni używać swojego tytułu.

Jak tu dotarliśmy?

Gdy powstawało pierwsze wydanie tej książki, przeglądarki Google Chrome oraz Mozilla Firefox były jeszcze stosunkowo młode. Wśród przeglądarek panowały niepodzielnie Internet Explorer 6 oraz Internet Explorer 7, a jego wersja 8 dopiero zaczynała zdobywać pewną popularność. Na szybki rozwój programowania w JavaScript wpływ miało kilka czynników.

Od momentu swego powstania JavaScript był zależny od przeglądarki. To właśnie przeglądarka była środowiskiem uruchomieniowym dla programów w tym języku, a dostęp programisty do funkcjonalności JavaScriptu był ściśle uzależniony od rodzaju oraz wersji przeglądarki używanej do wyświetlenia tworzonej przez niego strony. Wojna przeglądarek — zapoczątkowana w latach dziewięćdziesiątych ubiegłego wieku i ciągnąca się aż do połowy pierwszej dekady XXI wieku — została w łatwy sposób wygrana przez Internet Explorera, co spowodowało zastój w rozwoju innych przeglądarek. Jedynie dwie z nich potrafiły utrzymać się na rynku, a mianowicie: Mozilla Firefox oraz Google Chrome. Pierwsza z nich była potomkiem Netscape'a, który był jedną z najwcześniejszych przeglądarek internetowych. Chrome miał za to wsparcie firmy Google, co wystarczyło mu do błyskawicznego zaistnienia na scenie.

Najważniejsze jednak było to, że obie te przeglądarki zawierały kilka decyzji projektowych ułatwiających rozwój języka JavaScipt. Pierwszą z nich była decyzja o wsparciu implementacji różnych standardów zalecanych przez konsorcjum World Wide Web (W3C). Niezależnie od tego, czy mieliśmy do czynienia z modelem DOM, obsługą zdarzeń czy Ajaksem, zarówno Chrome, jak i Firefox w dużym stopniu uwzględniały specyfikację W3C oraz starały się implementować ją najlepiej, jak to było możliwe. Dla programistów oznaczało to, że nie musieli tworzyć oddzielnego kodu dla Firefoxa i oddzielnego dla Chrome'a. Już wcześniej przywykliśmy do tworzenia oddzielnego kodu dla IE oraz oddzielnego dla wszystkich innych przeglądarek, co sprawiało, że rozgałęzianie kodu nie było samo w sobie niczym nowym. Jednak to, że nie musiało być ono zbyt złożone, zdecydowanie ułatwiało pracę.

Twórcy zarówno Firefoksa, jak i Chrome'a przyczynili się w dużym stopniu do rozwoju Europejskiego Stowarzyszenia Wytwórców Komputerów (*European Computer Manufacturer Association*, w skrócie ECMA[1]). ECMA jest ciałem standaryzującym, które nadzoruje rozwój języka JavaScript (będąc skrupulatnym, należy wspomnieć, że ECMA nadzoruje język ECMAScript, ponieważ JavaScript jest znakiem handlowym firmy Oracle i… no cóż, tak naprawdę nie przejmujemy się takimi szczegółami, nieprawdaż? Dlatego w momencie odwoływania się do języka będziemy używać określenia JavaScript, zaś ECMAScript będzie używany do odwoływania się do specyfikacji, którą spełnia JavaScript). Wcześniej rozwój standardu ECMAScript popadł w marazm, podobnie jak miało to miejsce z rozwojem przeglądarki IE. Gdy rozpoczęła się prawdziwa rywalizacja przeglądarek, zaczęto również ponownie rozwijać standard ECMAScript. Wersja piąta tego standardu (z roku 2009) zatwierdzała wiele zmian, które powstały w ciągu dziesięciu poprzednich lat (!), czyli od czasu opublikowania poprzedniej wersji standardu. Sama grupa również nabrała sił, czego efektem było powstanie wersji 5.1 już w roku 2011. Zabezpieczona została również przyszłość standardu, ponieważ trwają już zaawansowane prace nad wersjami 6 oraz 7.

Twórcy Chrome'a również przyczynili się do uaktualnienia JavaScriptu. Silnik tego języka o nazwie V8 używany przez przeglądarkę Chrome stanowił istotną część debiutu tej przeglądarki w roku 2008. Zespół pracujący nad przeglądarką stworzył silnik zdecydowanie szybszy od większości innych silników języka JavaScript i postanowił utrzymać się na szczycie rankingu również w przypadku kolejnych wersji. Okazało się wkrótce, że silnik V8 był tak imponujący, że stał się podstawą narzędzia Node.js, które jest interpreterem języka JavaScript niezależnym od przeglądarki. Początkowo Node.js zaprojektowany

[1] Obecnie *European Association for Standardizing Information and Communication Systems — przyp. tłum.*

był jako serwer, który miał używać JavaScriptu jako swojego podstawowego języka aplikacji. Stał się jednak uniwersalną platformą do uruchamiania wielu aplikacji stworzonych w JavaScripcie.

Google Chrome wprowadził również inną dużą innowację do świata przeglądarek. Był to pomysł wiecznie aktualnej aplikacji. Zamiast zmuszać użytkownika do pobierania oddzielnego instalatora przeglądarki w celu jej uaktualnienia, Chrome domyślnie uaktualnia się sam w sposób zupełnie automatyczny. Chociaż takie podejście może sprawiać niekiedy problemy w świecie korporacji, to dla niezawodowego użytkownika przeglądarki stanowi prawdziwe dobrodziejstwo. Przeglądarka Chrome (a w ostatnich latach również Firefox) nadal uaktualnia się sama bez żadnego wysiłku ze strony użytkownika. Chociaż podobne podejście w przypadku uaktualnień Windows Update stosowała już od dłuższego czasu również firma Microsoft, to jednak nie wprowadzała w ten sposób żadnych nowych funkcjonalności przeglądarki Internet Explorer (chyba że było to związane z opublikowaniem nowej wersji systemu Windows). Inaczej mówiąc, uaktualnienia do przeglądarki IE nie pojawiają się zbyt często. To Chrome wraz z Firefoksem zawsze miały najnowsze i najlepsze funkcjonalności, a także były całkiem dobrze zabezpieczone.

Gdy firma Google zaczęła wprowadzać coraz więcej nowych funkcjonalności Chrome'a, twórcy innych przeglądarek też nie próżnowali. Niekiedy wynikały z tego zabawne sytuacje, np. wtedy gdy Firefox przyjął sposób numeracji używany przez Chrome'a. Firmy Mozilla oraz Microsoft przyjrzały się dokładnie silnikom języka JavaScript i stwierdziły, że choć przewaga Chrome'a wciąż jeszcze robi wrażenie, to jednak nie jest on nie do pokonania.

Ostatecznie firma Microsoft (w większości) porzuciła swoją filozofię „obejmij i rozszerz", przynajmniej w zakresie języka JavaScript. Równocześnie z wersją 9 przeglądarki IE także Microsoft zaimplementował obsługę zdarzeń rekomendowaną przez konsorcjum World Wide Web (W3C), a także ustandaryzował swoje interfejsy DOM i API Ajaksa. W przypadku większości standardowych funkcjonalności języka JavaScript nie ma już konieczności implementacji dwóch wersji tego samego kodu (pewnym problemem jest jeszcze tylko utrzymywanie starego kodu dla starych przeglądarek).

Wszystko to razem wygląda prawie na cudowne panaceum. JavaScript jest szybszy niż kiedykolwiek wcześniej. Zdecydowanie łatwiej jest tworzyć kod dla szeregu różnych przeglądarek. Dokumenty standaryzujące zarówno opisują rzeczywisty stan rzeczy, jak i wskazują wygodną drogę do kolejnych przyszłych funkcjonalności. Większość naszych przeglądarek jest w pełni aktualnych. Czym więc musimy się teraz martwić i dokąd będziemy zmierzać w przyszłości?

Współczesny JavaScript

Nigdy jeszcze tworzenie poważnych aplikacji za pomocą JavaScriptu nie było takie proste jak obecnie. Teraz można już zapomnieć o złych starych czasach, kiedy to konieczne było tworzenie oddzielnego kodu dla różnych przeglądarek, gdy w słaby sposób zaimplementowane były słabe standardy, a silniki JavaScript nie potrafiły szybko pracować. Przyjrzyjmy się obecnemu stanowi nowoczesnych środowisk JavaScript. W szczególności popatrzmy na dwa obszary: na nowoczesne przeglądarki oraz nowoczesne zestawy narzędzi.

Nowoczesny JavaScript opiera się na pomyśle nowoczesnej przeglądarki. Czym jest więc nowoczesna przeglądarka? Różne organizacje opisują to na różne sposoby. Google twierdzi, że jego aplikacje wspierane są przez obecne oraz poprzednie główne wersje przeglądarek (wiele wskazuje na to, że Gmail wciąż działa na IE9). Osoby reprezentujące firmę tworzącą stronę BBC (*British Broadcasting Company*) ujawniły w bardzo ciekawym artykule, że definiują następujące funkcjonalności reprezentujące nowoczesną przeglądarkę:

1. `document.querySelector()` / `document.querySelectorAll()`
2. `window.addEventListener()`
3. API dla składowania danych (`localStorage` and `sessionStorage`)

Najpopularniejszą biblioteką JavaScript w sieci jest biblioteka jQuery, która rozróżnia dwie wersje rozwojowe: serię 1.x wspierającą przeglądarki IE 6 oraz późniejsze oraz serię 2.x wspierającą „nowoczesne" przeglądarki takie jak IE 9 oraz wersje późniejsze. Niech Cię nie zmyli, drogi czytelniku, że IE jest granicą pomiędzy nowoczesnością a przeszłością. Pozostałe dwie główne linie przeglądarek są zawsze aktualne. Wprawdzie Safari oraz Opera nie mają aż tak dużego udziału w rynku co IE i nie są uaktualniane na bieżąco, ale jednak dzieje się to częściej niż w przypadku IE.

Gdzie więc leży granica oddzielająca nowoczesne przeglądarki od starszych typów? Płynna granica między nimi wydaje się znajdować pomiędzy dziewiątą a jedenastą wersją Internet Explorera. Internet Explorer w wersji 9 jest zdecydowanie przeglądarką starego typu. Nie wspiera on większości nowych funkcjonalności ECMAScript 5. Nie dysponuje również API do obsługi zdarzeń zgodnej z zaleceniami W3C. Ta lista może stale się wydłużać. Podczas omawiania nowoczesnych przeglądarek będziemy więc odwoływać się do Internet Explorera przynajmniej w wersji 9 i nie będziemy starać się wspierać starszych przeglądarek. Wszędzie tam, gdzie będzie to istotne i proste, postaramy się wskazać adaptację dla starszych wersji IE, jednak w ogólnym założeniu naszą granicą w tym zakresie będzie Internet Explorer 9.

Rozkwit bibliotek

Oprócz tematu nowoczesnych przeglądarek jest jeszcze jeden ważny aspekt obecnego środowiska dla języka JavaScript, o którym musimy wspomnieć: biblioteki. W ciągu ostatnich ośmiu lat nastąpiła eksplozja bibliotek JavaScript pod względem ich liczby oraz różnorodności. Na serwerze GitHub istnieje ponad 800 000 repozytoriów dla JavaScriptu. Spośród nich prawie 900 dysponuje tysiącem gwiazdek. Od czasu swoich skromnych początków jako zbioru funkcji narzędziowych ekosystem bibliotek JavaScript bardzo (choć nieco chaotycznie) się rozwinął.

W jaki sposób wpływa to na programistów JavaScriptu? Istnieje oczywiście model „biblioteki w charakterze rozszerzenia", w którym biblioteka udostępnia dodatkowe funkcjonalności. Są biblioteki MVC w rodzaju Backbone lub Angular (któremu przyjrzymy się w kolejnym rozdziale), biblioteki do wizualizacji danych jak d3 oraz Highcharts. Biblioteki JavaScriptu mogą również udostępniać pewien interfejs do funkcjonalności, które w niektórych przeglądarkach są standardowe, a w innych w ogóle nie występują.

Przez długi czas standardowym przykładem inaczej zaimplementowanej funkcjonalności JavaScriptu była obsługa zdarzeń. Internet Explorer dysponuje własnym API do obsługi zdarzeń. Inne przeglądarki zazwyczaj podążały za API rekomendowanym przez W3C. Wiele bibliotek udostępniało ujednolicone implementacje obsługi zdarzeń, które łączyły najlepsze cechy z obu światów. Niektóre z nich się nie rozwinęły, jednak te najbardziej popularne ujednoliciły funkcjonalność dla Ajaksa, modelu DOM oraz szeregu innych cech, które były w odmienny sposób zaimplementowane w różnych przeglądarkach.

Do najbardziej popularnych należała biblioteka jQuery. Od czasu swojego powstania biblioteka ta była słusznym wyborem w przypadku korzystania z nowych funkcjonalności JavaScriptu, gdyż nie trzeba było się martwić o to, czy będą one wspierane przez przeglądarkę. Dlatego zamiast używać obsługi zdarzeń w stylu IE lub W3C, można było skorzystać po prostu z funkcji jQuery `.on()`, która obsługiwała różnice, udostępniając ujednolicony interfejs. Kilka innych bibliotek również udostępniało podobną funkcjonalność. Wspomnieć należy o Dojo, Ext JS, Prototype, YUI, MooTools i wielu innych. Celem tych bibliotek narzędziowych było ujednolicenie API dla programistów.

Standaryzacja wykroczyła nawet poza udostępnianie prostego kodu działającego na różnych platformach. Wspomniane biblioteki bardzo często ukrywały nawet błędnie działające implementacje. Oficjalne API dla funkcji pomiędzy różnymi wersjami przeglądarki może nawet się nie zmienić w dużym stopniu, ale xi tak zawsze będą występować w nim błędy. Niekiedy błędy te zostaną naprawione, innym razem nie, a niekiedy poprawki spowodują wprowadzenie nowych błędów. Wszędzie tam, gdzie możliwe było naprawienie lub obejście tych błędów, biblioteki robiły to. Na przykład biblioteka jQuery 1.11 zawiera ponad pół tuzina poprawek dla problemów z API obsługi zdarzeń.

Niektóre biblioteki (w szczególności jQuery) udostępniały również nowe lub inne implementacje pewnych właściwości. Na przykład funkcja selektora jQuery (tworząca rdzeń biblioteki) wyprzedzała

będące obecnie standardem funkcje `querySelector()` oraz `querySelectorAll()`. Stanowiła również powód dołączenia tych funkcji do języka JavaScript. Inne biblioteki udostępniały różne funkcjonalności pomimo ich bardzo różnych implementacji. W dalszej części tej książki przyjrzymy się nowemu protokołowi Ajaksa o nazwie CORS (Cross Origin Resource Sharing). Pozwala on Ajaksowi wysyłać żądania do innych serwerów niż te, z których obsługiwana była strona. Niektóre biblioteki zaimplementowały już taką funkcjonalność, jednak wszędzie tam, gdzie było to konieczne, korzystały z protokołu JSON-P.

Z powodu swojej przydatności niektóre biblioteki stały się częścią składową standardowego zestawu narzędzi zawodowego programisty JavaScript. Ich funkcje nie muszą być jeszcze ustandaryzowane w JavaScripcie, stanowią jednak zbiór wiedzy oraz funkcjonalności, które w prosty sposób ułatwiają szybką realizację niektórych wzorców. Okazało się, że jQuery (lub inna podobna biblioteka) jest właściwie niezbędna do stworzenia nowoczesnej przeglądarki, dzięki czemu w ubiegłych latach można było zwiększyć popularność swojego bloga. Wróćmy do wymagań BBC — z pewnością można by było zrealizować dużą część funkcjonalności udostępnianej przez jQuery, gdyby dostępne były trzy wymienione wtedy metody. Biblioteka jQuery oferuje również uproszczony (ale wciąż jeszcze rozszerzony) interfejs DOM, poprawiając błędy dla wielu różnego rodzaju przypadków brzegowych. Jeżeli potrzebujesz wsparcia przeglądarki IE8 lub wcześniejszej, jQuery będzie Twoim podstawowym wyborem. Jednocześnie zawodowy programista JavaScript musi przyjrzeć się wymaganiom projektu i zastanowić się, czy podejmie ryzyko odkrywania ponownie czegoś, co już udostępnia jQuery (lub podobna biblioteka).

Więcej niż wzmianka o rozwiązaniach mobilnych

W starszych książkach na temat JavaScriptu lub programowania stron internetowych ujrzałbyś prawdopodobnie sekcję, a być może i cały rozdział dotyczący postępowania w przypadku przeglądania stron w trybie mobilnym. Udostępnianie stron w tym trybie stanowiło jedynie mały fragment całkowitego przeglądania sieci. Rynek był tak podzielony, że wydawało się, iż jedynie specjaliści mogliby być zainteresowani tworzeniem rozwiązań mobilnych. Nie jest to już na szczęście prawdą. Od czasu pierwszego wydania tej książki rynek usług mobilnych eksplodował. Różni się on też zdecydowanie od programowania na komputery stacjonarne. Rozważmy pewne statystyki: według wielu różnych źródeł przeglądanie stron z urządzeń mobilnych stanowi 20 – 30 procent wszystkich odwołań. Gdy będziesz, drogi Czytelniku, czytać te słowa, zapewne procent ten już się zwiększy, ponieważ od czasu debiutu iPhone'a ten udział ulega stałemu zwiększeniu. Znacznie ponad 40 procent odwołań do stron z urządzeń mobilnych pochodzi z przeglądarki Safari pod systemem iOS, chociaż przeglądarka Androida oraz Chrome dla tego systemu również umacniają swoją pozycję (i w zależności od tego, jakim danym statystycznym wierzysz, mogły już nawet przegonić Safari). Przeglądarka Safari w systemie iOS nie jest tym samym, co Safari na komputerze stacjonarnym i taka sama różnica występuje pomiędzy przeglądarkami Chrome oraz Firefox na urządzeniach mobilnych i stacjonarnych. Urządzenia mobilne dominują.

Przeglądarki na urządzenia mobilne wyznaczają zupełnie nowy zestaw wyzwań oraz możliwości. Urządzenia mobilne są często o wiele bardziej ograniczone niż stacjonarne (chociaż jest to kolejna różnica, która w szybkim tempie maleje). Z drugiej strony urządzenia mobilne oferują nowe funkcjonalności (zdarzenia przeciągnięcia, bardziej dokładne dane geolokacyjne itp.) oraz nowe sposoby interakcji (wykorzystanie dłoni zamiast myszki, przeciąganie w celu przewinięcia stron). W zależności od wymagań w stosunku do programu możesz utworzyć aplikację wyglądającą równie dobrze na urządzeniu mobilnym, co na stacjonarnym lub też zaimplementować ponownie istniejący zakres funkcjonalności na platformie mobilnej.

Scena języka JavaScript w ciągu ostatnich lat uległa zdecydowanej zmianie. Poza pewnymi elementami standaryzacji API powstało wiele nowych wyzwań. W jaki sposób to wszystko może dotykać nas, czyli zawodowych programistów JavaScript?

Dokąd zmierzamy?

Powinniśmy ustanowić dla siebie samych pewne standardy. Jednym z nich jest to, że IE 9 to dolna granica dla nowoczesnych przeglądarek. Inne przeglądarki są zawsze aktualne i nie musimy się o nic martwić. A co w takim razie z przeglądarkami mobilnymi? No cóż, jest to bardziej złożone zagadnienie. Uogólniając, minimalne wersje to systemy iOS 6 oraz Android 4.1 (Jelly Bean). Wersje mobilne zazwyczaj są uaktualniane szybciej i częściej niż ich odpowiedniki stacjonarne, dlatego też możemy używać nowszych wersji mobilnych systemów operacyjnych z większym przekonaniem.

Zboczmy na chwilę z tematu, pomińmy dyskusję o wersjach przeglądarek, systemów operacyjnych lub platform i skoncentrujmy się na użytkownikach. Przytaczanie statystyk może trwać nawet cały dzień, jednak te wartościowe mówią coś ważnego o Twoich użytkownikach, a nie o wszystkich użytkownikach ogólnie. Być może tworzysz aplikację dla swojego pracodawcy, który używa standardu IE 10. A może Twój pomysł na aplikację zależy zdecydowanie od funkcjonalności oferowanych jedynie w przeglądarce Chrome. Może nawet nie powstanie wersja na komputery stacjonarne, gdyż zamierzasz używać jedynie tabletów z systemem Android lub iPadów. Zastanów się nad użytkownikami docelowymi. Ta książka jest napisana w taki sposób, aby miała dość szerokie zastosowanie. To samo może dotyczyć aplikacji. Nierozsądne więc wydaje się tracenie czasu na zastanawianie się nad błędami w IE 9, jeśli chodzi o aplikacje tylko na tablety. Wróćmy teraz do naszych standardów.

W przypadku zrzutów ekranów lub testów w książce będziemy zazwyczaj korzystać z przeglądarki Google Chrome. Jeśli zaistnieje taka potrzeba, będziemy demonstrować kod w przeglądarce Firefox lub Internet Explorer. Chrome jest złotym standardem dla programistów z powodu serwowanych im informacji, jest też przyjazny dla użytkowników. W kolejnym rozdziale poznamy wiele dostępnych narzędzi programistycznych oferowanych nie tylko przez Chrome'a, ale także przez Firefoxa (z wtyczką Firebug lub bez niej) oraz IE.

Jako standardowej biblioteki będziemy używać jQuery. Oczywiście istnieje wiele innych bibliotek, lecz jQuery zwycięża z dwóch powodów. Po pierwsze, jest najpopularniejszą biblioteką JavaScript ogólnego przeznaczenia dostępną w sieci. Po drugie, jeden z autorów tej książki (John Resig) zna dobrze bibliotekę jQuery, co spowodowało, że drugi autor (John Paxton) dał za wygraną. Przygotowując drugą wersję tej książki, wiele technik z poprzedniego wydania zastąpiliśmy funkcjonalnością dostępną w bibliotece jQuery. W takich przykładach dość niechętnie odkrywamy koło na nowo. W razie potrzeby odwołamy się do odpowiedniej funkcjonalności jQuery. Będziemy oczywiście omawiać także nowe, ekscytujące techniki.

W ciągu ostatnich kilku lat IDE JavaScriptu znacząco się poprawiło, głównie dzięki rozwojowi języka. Lista wszystkich możliwości jest zbyt liczna, aby je wszystkie wymieniać w tym miejscu, jeżeli jednak mielibyśmy wymieniać kilka znaczących aplikacji, to John Resig używa w swoim środowisku programistycznym bardzo zindywidualizowanej wersji edytora vim. John Paxton jest nieco bardziej leniwy i postanowił używać w charakterze swojego IDE doskonałego edytora WebStorm firmy JetBrains (*http://www.jetbrains.com/webstorm*). Firma Adobe oferuje jako bezpłatne oprogramowanie open source edytora Brackets (*http://brackets.io/*), mającego obecnie wersję 1.3. Dostępna jest również wersja edytora Eclipse. Wiele osób z kolei bardzo pozytywnie ocenia dostosowanie do swoich potrzeb edytorów SublimeText lub Emacs. Ty natomiast używaj tego, z czym czujesz się najswobodniej.

Istnieją również inne narzędzia, które mogą wspomagać programowanie w języku JavaScript. Poznamy je w jednym z dalszych rozdziałów. Oznacza to, że nadszedł dobry czas na zaprezentowanie tego, co nas będzie czekać w dalszej części książki.

Co nas czeka wkrótce?

Począwszy od rozdziału 2., rozpoczniemy naszą przygodę z najnowszą i najlepszą wersją języka JavaScript. Będziemy analizować nowe funkcjonalności dostępne dzięki typowi `Object`. Zbadamy też ponownie niektóre stare pomysły, takie jak referencje, funkcje, zakres oraz domknięcia. Zgrupujemy to wszystko pod nazwą **Funkcji, Cech oraz Obiektów**, jednak rozdział ten będzie zawierać znacznie więcej informacji.

W rozdziale 3. omówimy **tworzenie kodu do wielokrotnego użytku**. W rozdziale 2. prześlizgniemy się jedynie po jednej z większych nowych cech języka JavaScript, jaką jest metoda `Object.create()`, poznamy także jej wpływ na obiektowy kod JavaScript. Dopiero więc w rozdziale trzecim poznamy dokładniej metodę `Object.cresate()`, funkcjonalne konstruktory, prototypy oraz zagadnienia obiektowe zaimplementowane w JavaScript.

Po poświęceniu dwóch rozdziałów na omówienie kodu czas zacząć myśleć, w jaki sposób nim zarządzać. W rozdziale 4. przedstawimy **narzędzia do testowania kodu w JavaScript**. Zaczniemy od sprawdzenia przeglądarek oraz ich narzędzi programistycznych.

Rozdział 5. rozpoczniemy od omówienia obszarów funkcjonalności JavaScript o wysokim stopniu użycia. Przyjrzymy się wtedy również **modelowi obiektowemu dokumentu** DOM. Od ostatniej wersji API DOM zdecydowanie stało się dużo bardziej złożone, co utrudnia jego zastosowanie. Istnieje jednak kilka nowych jego cech, które powinniśmy poznać.

W rozdziale 6. postaramy się opanować zdarzenia (*Events*), które zostały poddane standaryzacji zgodnie z zaleceniami konsorcjum W3C. Pozwoli to nam na ostateczne odejście od bibliotek narzędziowych i zagłębienie się w API zdarzeń bez troszczenia się o duże różnice pomiędzy przeglądarkami.

Jednym z pierwszych (innych niż rozrywkowe) zastosowań JavaScript była walidacja formularzy po stronie klienta. Twórcom przeglądarek zajęło ponad dekadę, aby zdecydować się na dodanie funkcjonalności służącej do walidacji formularzy poza prostym przechwytywaniem zdarzenia `submit`. W rozdziale 7. pt. **JavaScript a walidacja formularzy** odkryjemy, że istnieje całkiem nowy zestaw funkcjonalności, służących do walidacji formularzy, udostępniany zarówno przez język HTML, jak i JavaScript.

Każdy, kto zajmował się programowaniem w języku JavaScript, musiał poświęcić nieco czasu na **Wprowadzenie do Ajaksa**. Od czasu wprowadzenia protokołu CORS funkcjonalność protokołu Ajax przezwyciężyła ostatecznie najgłupsze ze swoich ograniczeń.

W rozdziale pt. **Narzędzia do tworzenia stron internetowych** zostały opisane narzędzia (takie jak Yeoman, Bower, Git lub Grunt), które są wywoływane z wiersza poleceń. Narzędzia te pokażą nam, w jaki sposób można szybko dodać potrzebne pliki oraz katalogi. Dzięki temu możemy skoncentrować się na programowaniu.

W rozdziale 10. omówimy **AngularJS oraz testowanie**. Korzystając z wiedzy nabytej w poprzednim rozdziale, dowiemy się, co sprawia, że Angular działa, a także w jaki sposób można zaimplementować zarówno testy jednostkowe, jak i całościowe.

W rozdziale 11. omówimy **Przyszłość języka JavaScript**. Do czasu wydania tej książki zostanie już mniej więcej ustanowiony standard ECMAScript w wersji 6. ECMAScript w wersji 7 wciąż aktywnie się rozwija. Po przedstawieniu planów rozwoju JavaScriptu zajmiemy się głównie tymi funkcjonalnościami, których można używać obecnie.

Podsumowanie

Dużą część tego rozdziału poświęciliśmy omawianiu otoczenia JavaScriptu: platformom, historii, IDE itp. Wierzymy, że historia determinuje teraźniejszość. Próbowaliśmy wyjaśnić, gdzie byliśmy przedtem i jak dostaliśmy się tu, gdzie jesteśmy teraz, aby pomóc Ci zrozumieć, gdzie obecnie znajduje się język JavaScipt i czym on właściwie jest. Jesteśmy przekonani, że ta książka omawia takie techniki oraz API, które powinien znać każdy zawodowy programista JavaScript. Tak więc bez dalszego zbytniego przedłużania…

Skorowidz

E

F

G

H

I

J

K

M

N

O

P

R

S

T

U

W

Y

Z

Ż

Zaawansowane techniki języka JavaScript

JavaScript stanowi doskonałe, zaawansowane narzędzie do programowania aplikacji internetowych. Zawodowe wykorzystanie tego języka wymaga jednak opanowania trudniejszych zagadnień, takich jak obiektowy model dokumentu (DOM) czy korzystanie z nowych narzędzi obiektowych w JS. Trzeba też nauczyć się testować aplikację i usuwać błędy. Dopiero wiedza na tym poziomie pozwoli zasłużyć na miano profesjonalisty.

Niniejsza książka stanowi kompendium wiedzy o nowoczesnym języku JavaScript. Zawiera zwięzłe, praktyczne informacje, których każdy doskonały programista będzie potrzebował do pisania aplikacji internetowych. Autorzy skoncentrowali się na zasadniczych tematach i przedstawili wszystkie istotne kwestie dotyczące zaawansowanych technik programowania w tym języku. Opisali również praktyczne sposoby wykorzystania tych technik, wskazując przy tym sposoby uniknięcia potencjalnych problemów.

Dzięki tej książce:

- poznasz najważniejsze profesjonalne techniki programowania w JavaScripcie
- zrozumiesz działanie funkcji i zapoznasz się z narzędziami obiektowymi tego języka
- dokładnie zgłębisz kwestie związane z modelem DOM i nauczysz się go wykorzystywać
- zapoznasz się z zasadami obsługi zdarzeń w JavaScripcie
- nauczysz się tworzyć kod wielokrotnego użytku z wykorzystaniem obiektowego JavaScriptu
- zaczniesz wykorzystywać JavaScript do walidacji formularzy HTML oraz CSS
- opanujesz techniki, które wkrótce zdecydują o przyszłości JavaScriptu

John Resig — twórca biblioteki JavaScript o nazwie jQuery. Autor książek poświęconych temu językowi, takich jak *Tajemnice JavaScriptu. Podręcznik Ninja* (Helion 2014). Prelegent na konferencjach organizowanych przez takie firmy jak Yahoo! czy Google.

Russ Ferguson — doświadczony programista i wykładowca. Autor aplikacji dla takich firm jak Morgan Stanley, Comcast, Chase Bank, Publicis Groupe, DC Comics czy MTV/Viacom.

John Paxton — programista, trener, autor i wykładowca. Posługuje się wieloma językami programowania, jednak obecnie koncentruje się głównie na JavaScripcie i Javie.

Helion
41998 numer katalogowy
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900
Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci
Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl
Informatyka w najlepszym wydaniu
Apress
ISBN 978-83-283-2086-4
9 788328 320864
cena: 44,90 zł