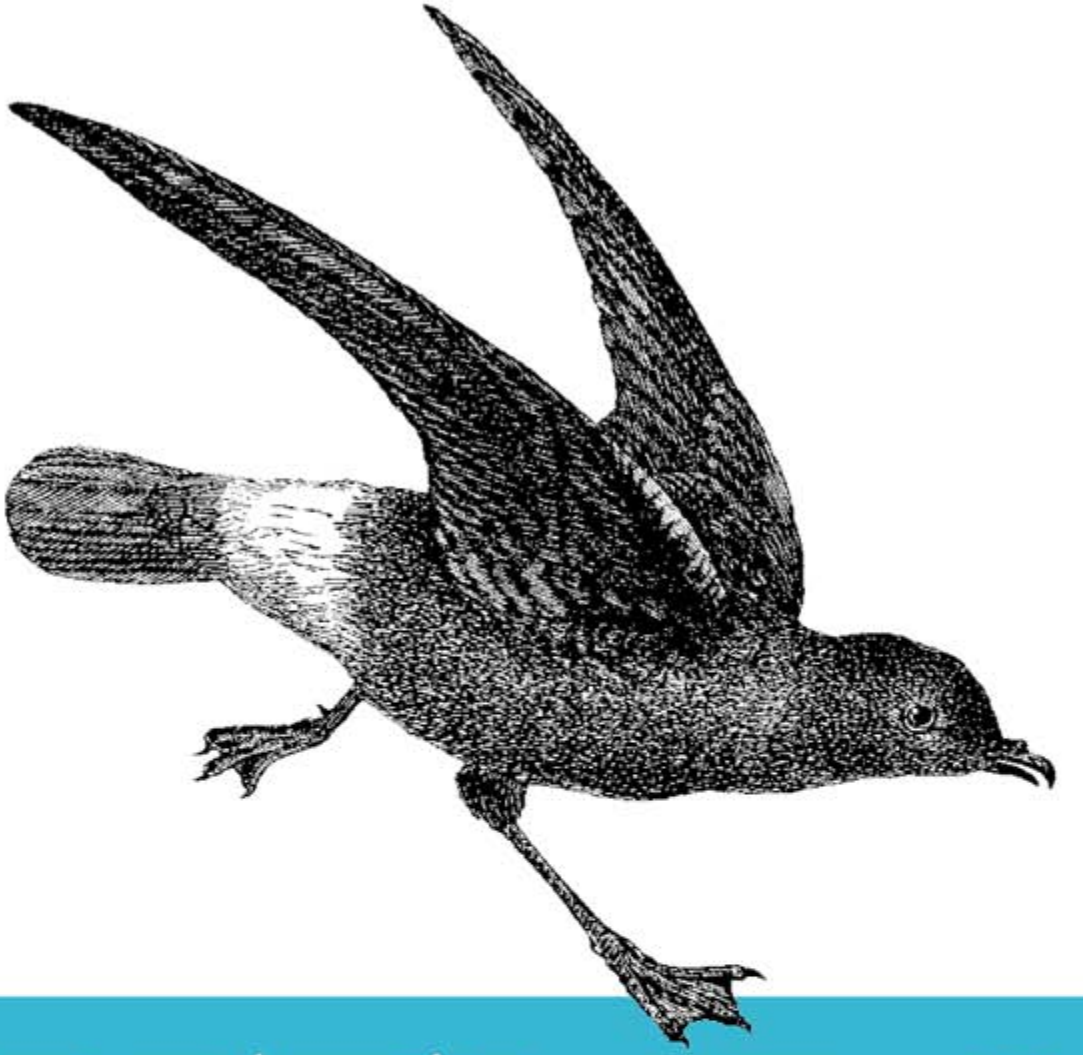
Twórz w pełni funkcjonalne rozwiązania dla wielu platform!



HTML5

Programowanie aplikacji

HELION

O'REILLY®

Zachary Kessin

Tytuł oryginalny: Programming HTML5 Applications

Tłumaczenie: Piotr Pilch

ISBN: 978-83-246-4897-9

© 2012 Helion S.A.

Authorized Polish translation of the English edition of Programming HTML5 Applications 1st Edition ISBN 9781449399085 © 2012 Zachary Kessin

This translation is published and sold by permission od O'Reilly Media, Inc., which owns or cotrols rights to publish and sell the same.

Polish edition copyright © 2012 by Helion S.A.
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/htm5pa.zip

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/ htm5pa
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Przedmowa **7**

1. Sieć WWW jako platforma aplikacji **11**
Zwiększanie możliwości aplikacji internetowych 12
Projektowanie aplikacji internetowych 13
Triumf języka JavaScript 15

2. Możliwości języka JavaScript **19**
Nieblokujące operacje wejścia-wyjścia i wywołania zwrotne 20
Funkcje lambda oferują duże możliwości 22
Domknięcia 24
Programowanie funkcyjne 27
Prototypy i sposób rozszerzania obiektów 30
Rozszerzanie funkcji przy użyciu prototypów 33
Rozwijanie i parametry obiektów 36
Operacje iteracji dotyczące tablicy 37
Obiekty również mogą być rozwijane 41

3. Testowanie aplikacji JavaScript **43**
QUnit 47
Selenium 50

4. Lokalne magazynowanie danych **71**
Obiekty localStorage i sessionStorage 73
Dodatki biblioteki jQuery 82

Kup książkę Poleć książkę

5. Interfejs IndexedDB 85
Dodawanie i aktualizowanie rekordów 89
Dodawanie indeksów 90
Pobieranie danych 91
Usuwanie danych 92

6. Pliki 93
Obiekty blob 94
Praca z plikami 95
Wysyłanie plików 97
Przeciąganie i upuszczanie 98
Połączenie wszystkiego ze sobą 99
System plików 101

7. Praca w trybie bez połączenia 103
Plik manifestu — wprowadzenie 104
Zdarzenia 108
Debugowanie plików manifestu 109

8. Podział pracy za pomocą technologii Web Workers 113
Przypadki zastosowania wątku roboczego Web Worker 115
Zastosowanie technologii Web Workers 117
Przykład fraktala bazującego na wątku roboczym 119
Testowanie i debugowanie wątków roboczych Web Worker 127
Wzorzec ponownego wykorzystania przetwarzania wielowątkowego 127
Biblioteki dla technologii Web Workers 132

9. Gniazda WWW 133
Interfejs gniazd WWW 135
Konfigurowanie gniazda WWW 136
Przykład gniazda WWW 136
Protokół gniazd WWW 139

Kup książkę Poleć książkę

10. Nowe znaczniki 143
Znaczniki dla aplikacji 143
Ułatwienie dostępu za pomocą aplikacji WAI-ARIA 145
Mikrodane 146
Nowe typy formularzy 147
Dźwięk i wideo 149
Element canvas i format SVG 149
Geolokacja 150
Nowy kod CSS 150

A Narzędzia JavaScript, które warto znać 153

Skorowidz 157

Kup książkę Poleć książkę

Kup książkę Poleć książkę

ROZDZIAŁ 4.

Lokalne magazynowanie danych

Przeglądarka internetowa oferuje użytkownikom skryptów JavaScript znakomite środowisko służące do budowania aplikacji uruchamianych w przeglądarce. Gdy używa się biblioteki ExtJS lub jQuery, możliwe jest utworzenie aplikacji, która w opinii wielu użytkowników może rywalizować z tym, co oferuje tradycyjna aplikacja. Ponadto można skorzystać z wyjątkowo prostej metody dystrybucji. Choć przeglądarka świetnie wypada pod względem komfortu obsługi interfejsu użytkownika, nie może się już jednak pochwalić tym samym w odniesieniu do magazynowania danych.

Dawniej przeglądarki nie potrafiły w żaden sposób magazynować danych. Praktycznie były one jedynym w swoim rodzaju „cienkim" klientem. To, co pojawiło się jako pierwsze, był to mechanizm obsługi informacji *cookie* protokołu HTTP, który umożliwia powiązanie porcji danych z każdym żądaniem HTTP. Jednakże w przypadku informacji *cookie* pojawia się kilka problemów. Po pierwsze, każda taka informacja jest odsyłana z każdym żądaniem. A zatem przeglądarka wysyła informację *cookie* dla każdego pliku JavaScript, obrazu, żądania Ajax itp. Bez żadnego uzasadnionego powodu może to spowodować znaczne wykorzystanie przepustowości połączenia. Po drugie, w specyfikacji informacji *cookie* tak ją zdefiniowano, aby mogła być współużytkowana w wielu różnych domenach. Jeśli do firmy należały domeny *app.test.com* i *images.test.com*, informacja *cookie* mogła zostać ustawiona dla obu jako widoczna. Problem polega na tym, że poza Stanami Zjednoczonymi powszechne stają się nazwy domen złożone z trzech części. Na przykład możliwe byłoby ustawienie informacji *cookie* dla wszystkich hostów w domenie *.co.il*. W efekcie informacja *cookie* mogłaby trafić do niemal każdego hosta w Izraelu. Ponadto nie jest możliwe wymaganie trzyczęściowej nazwy domeny każdorazowo, gdy nazwa zawiera przyrostek kraju, ponieważ niektóre państwa, takie jak Kanada, stosują inną konwencję.

Kup książkę Poleć książkę

Możliwość lokalnego magazynowania w przeglądarce zapewnia poważną korzyść pod względem szybkości. W zależności od serwera wykonywanie normalnego zapytania Ajax może zająć od pół sekundy do kilku sekund. Jednakże nawet w najlepszym możliwym przypadku może to być dość długi czas. Przesłanie prostego żądania narzędzia ping protokołu ICMP z mojego biura w Tel Awiwie do serwera w Kalifornii zajmie średnio 250 ms. Duża część tego czasu będzie prawdopodobnie wynikiem podstawowych ograniczeń fizycznych: dane mogą być przesyłane w kablu tylko z określoną częścią ułamkową prędkości światła. A zatem bardzo niewiele można zrobić w celu zwiększenia szybkości, dopóki dane muszą pokonywać drogę z przeglądarki do serwera.

Opcje lokalne magazynowania bardzo dobrze się sprawdzają w przypadku danych, które są statyczne lub głównie statyczne. Na przykład wiele aplikacji zawiera listę państw jako część swoich danych. Jeśli nawet lista uwzględnia kilka dodatkowych informacji, takich jak to, czy produkt jest oferowany w każdym kraju, nie będzie się zmieniać zbyt często. W tym przypadku nierzadko sprawdza się rozwiązanie polegające na wstępnym załadowaniu danych do obiektu `localStorage`, a następnie w razie potrzeby warunkowym ponownym ładowaniu, aby użytkownik uzyskał wszelkie nowe dane bez konieczności oczekiwania na te bieżące.

Oczywiście lokalne magazynowanie jest również istotne podczas pracy z aplikacją internetową, która może działać w trybie bez połączenia. Choć obecnie dostęp do internetu może wydawać się wszechobecny, nie powinien być traktowany jako coś uniwersalnego (nawet w inteligentnych telefonach). Użytkownicy z takimi urządzeniami jak iPod uzyskają dostęp do internetu tylko wtedy, gdy istnieje połączenie WiFi. Nawet w przypadku inteligentnych telefonów, takich jak iPhone lub Android, wystąpią „martwe" strefy bez dostępu do sieci.

Wraz z rozwojem języka HTML5 dokonano sporego postępu w zakresie zapewniania przeglądarce metody tworzenia trwałego magazynu lokalnego. Jednakże wyniki tych prac wymagają jeszcze nabrania realnych kształtów. Obecnie istnieją co najmniej trzy propozycje dotyczące sposobu przechowywania danych w kliencie.

W roku 2007 w ramach projektu Gears firma Google zaprezentowała bazę danych SQLite opartą na przeglądarce. W przeglądarkach bazujących na mechanizmie WebKit, w tym Chrome, Safari oraz przeglądarkach dla telefonów iPhone i Android, zaimplementowano wersję bazy danych Gears SQLite. Jednak tę bazę danych usunięto z propozycji języka HTML5, ponieważ jest ona komponentem z jednym źródłem.

Kup książkę Poleć książkę

Mechanizm `localStorage` udostępnia obiekt JavaScript, który jest zachowywany w kolejnych operacjach ponownego ładowania stron internetowych. Ten mechanizm wydaje się dość dobrze uzgodniony i stabilny. Jest odpowiedni do przechowywania danych o niewielkim rozmiarze, takich jak informacje o sesji lub preferencje użytkownika.

W tym rozdziale wyjaśniono, jak używać implementacji mechanizmu `localStorage`. W rozdziale 5. zajmiemy się bardziej skomplikowaną i zaawansowaną odmianą lokalnego magazynowania, która pojawiła się w niektórych przeglądarkach. Mowa o interfejsie IndexedDB.

Obiekty localStorage i sessionStorage

Na potrzeby magazynowania nowoczesne przeglądarki oferują programiście dwa obiekty `localStorage` i `sessionStorage`. Każdy z tych obiektów może przechowywać dane jako klucze i wartości. Obiekty mają ten sam interfejs i działają jednakowo z jednym wyjątkiem. Obiekt `localStorage` jest zachowywany w kolejnych uruchomieniach przeglądarki, natomiast obiekt `sessionStorage` jest resetowany w chwili ponownego uruchomienia sesji przeglądarki. Może to mieć miejsce w przypadku zamykania przeglądarki lub okna. Dokładny moment wystąpienia tego zdarzenia będzie zależny od szczegółów przeglądarki.

Ustawianie tych obiektów i uzyskiwanie do nich dostępu jest naprawdę proste (przykład 4.1).

Przykład 4.1. Uzyskiwanie dostępu do obiektu localStorage

```
// Ustawianie
localStorage.sessionID = sessionId;
localStorage.setItem('sessionID', sessionId);

// Uzyskiwanie dostępu

var sessionId;
sessionId = localStorage.sessionID;
sessionId = localStorage.getItem('sessionId');

localStorage.sessionId = undefined;
localStorage.removeItem('sessionId');
```

Dane magazynowane przez przeglądarkę, takie jak informacje *cookie*, implementują omawianą zasadę „tego samego źródła”. Z tego powodu różne witryny internetowe nie mogą kolidować z inną witryną lub odczytywać jej danych. Jednakże oba obiekty opisywane w tym podrozdziale są

Kup książkę Poleć książkę

przechowywane na dysku użytkownika (tak jak informacje *cookie*), dlatego bardziej zaawansowany użytkownik może znaleźć sposób na edytowanie danych. Narzędzia Developer Tools przeglądarki Chrome umożliwiają programiście edytowanie obiektu magazynowania. Taki obiekt może być również edytowany w przeglądarce Firefox za pośrednictwem dodatku Firebug lub innego narzędzia. Oznacza to, że choć inne witryny nie mogą uzyskać dostępu do danych w obiektach magazynowania, nie powinny one jednak być godne zaufania.

Informacji *cookie* dotyczą określone restrykcje. Ich maksymalna wielkość to około 4 kB. Ponadto informacje te muszą być przesyłane do serwera z każdym żądaniem Ajax, co znacznie zwiększa ruch w sieci. Obiekt `localStorage` przeglądarki pozwala na o wiele więcej. Choć specyfikacja języka HTML5 nie określa dokładnego limitu wielkości obiektu, w większości przeglądarek na jednego hosta internetowego jest ustalony taki limit i wynosi on około 5 MB. Programista nie powinien przyjmować bardzo dużego obszaru magazynowania.

Dane mogą być przechowywane w obiekcie magazynowania z wykorzystaniem bezpośredniego dostępu do niego lub przy użyciu zestawu funkcji dostępu. Obiekt sesji może przechowywać wyłącznie łańcuchy, dlatego każdy magazynowany obiekt będzie rzutowany na łańcuch. Oznacza to, że obiekt będzie przechowywany w postaci `[object Object]`, która prawdopodobnie nie jest tym, czego się oczekuje. Aby przechowywać obiekt lub tablicę, należy je najpierw przekształcić do formatu JSON.

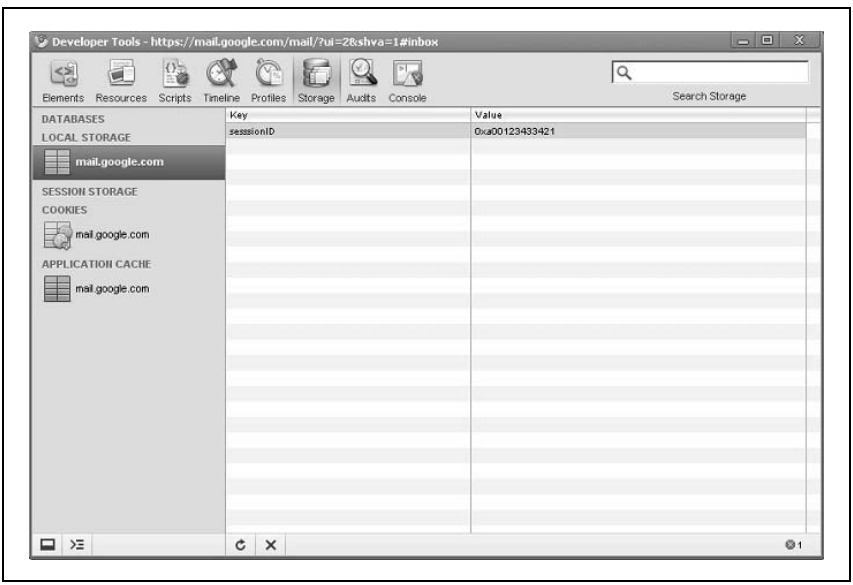
Zmiana wartości w obiekcie magazynowania każdorazowo powoduje wywołanie zdarzenia magazynowania. To zdarzenie wyświetli klucz, a także jego poprzednią i nową wartość. W przykładzie 4.2 zaprezentowano typową strukturę danych. W przeciwieństwie do niektórych zdarzeń, takich jak kliknięcia, zdarzenia magazynowania nie mogą zostać wyeliminowane. Aplikacja nie ma możliwości poinformowania przeglądarki o tym, aby nie wprowadzała zmiany. Zdarzenie po prostu informuje aplikację o zmianie już po fakcie.

Przykład 4.2. Interfejs zdarzenia magazynowania

```
var storageEvent = {
    key: 'key',
    oldValue: 'old',
    newValue: 'newValue',
    url: 'url',
    storageArea: storage // Zmodyfikowany obszar magazynowania
};
```


Kup książkę Poleć książkę

Mechanizm WebKit udostępnia okno w ramach własnych narzędzi Developer Tools, w którym programista może wyświetlić i edytować obiekty `localStorage` i `sessionStorage` (rysunek 4.1). W oknie należy kliknąć kartę *Storage*. Spowoduje to wyświetlenie obiektów `localStorage` i `sessionStorage` na stronie. Okno oferuje też pełne możliwości edycji. Klucze mogą być w nim dodawane, usuwane i modyfikowane.



Rysunek 4.1. Przeglądarka obiektów magazynowania aplikacji Chrome

Choć w przeciwieństwie do przeglądarki Chrome oraz innych przeglądarek opartych na mechanizmie WebKit dodatek Firebug nie udostępnia interfejsu do obiektów `localStorage` i `sessionStorage`, dostęp do nich może być uzyskany za pośrednictwem konsoli JavaScript. Umożliwia ona dodawanie, edytowanie i usuwanie kluczy. Spodziewam się, że kiedyś ktoś utworzy rozszerzenie dodatku Firebug, które na to pozwoli.

Oczywiście możliwe jest utworzenie interfejsu niestandardowego, służącego do wyświetlania i edytowania obiektów magazynowania w dowolnej przeglądarce. Należy utworzyć widget wyświetlany na ekranie, który prezentuje obiekty przy użyciu wywołań metod `getItem` i `removeItem` (przykład 4.1), i zezwolić na edycję za pomocą pól tekstowych. Szkielet widgetu zaprezentowano w przykładzie 4.3.

Kup książkę Poleć książkę

Przykład 4.3. Przeglądarka obiektów magazynowania

```
(function createLocalStorageViewer()
{
  $('<table></table>').attr(
  {
    "id": "LocalStorageViewer",
    "class": 'hidden viewer'
  }).appendTo('body');
  localStorage.on('update', viewer.load);
var viewer =
  {
    load: function loadData()
    {
      var data, buffer;
      var renderLine = function (line)
      {
        return "<tr key='{key}' value='{value}'>\n".populate(line) +
          "<td class='remove'>Usuń klucz</td>" +
          "<td class='storage-key'>{key}</td><td>{value}</td></tr>".populate(line);
      };
      buffer = Object.keys(localStorage).map(function (key)
      var rec =
      {
        key: key,
        value: localStorage[data]
      };
      return rec;
      });
  };
  $("#LocalStorageViewer").html(buffer.map(renderLine).join(''));
  $("#LocalStorageViewer tr.remove").click(function ()
  {
    var key = $(this).parent('tr').attr('key').remove();
    localStorage[key] = undefined;
  });
  $("#LocalStroageViewer tr").dblclick(function ()
  {
    var key = $(this).attr('key');
    var value = $(this).attr('value');
    var newValue = prompt("Czy zmienić wartość " + key + "?", value);
    if (newValue !== null)
    {
      localStorage[key] = newValue;
    }
  });
};
}());
```


Kup książkę Poleć książkę

Użycie obiektu localStorage w bibliotece ExtJS

Biblioteka ExtJS, której kilka przykładów zamieszczono w poprzednich rozdziałach, to niezwykle popularna struktura języka JavaScript, umożliwiająca bardzo zaawansowane, interaktywne operacje wyświetlania. W tym punkcie wyjaśniono, jak używać obiektu `localStorage` z biblioteką ExtJS.

Wartościową funkcją tej biblioteki jest to, że wiele jej obiektów może zapamiętywać własny stan. Na przykład obiekt siatki biblioteki ExtJS umożliwia użytkownikowi zmianę wielkości, ukrywanie i wyświetlanie oraz zmianę kolejności kolumn. Te zmiany są zapamiętywane i ponownie wyświetlane, gdy użytkownik powróci później do aplikacji. Pozwala to każdemu użytkownikowi dostosować sposób działania elementów aplikacji.

Biblioteka ExtJS udostępnia obiekt do zapisu stanu, lecz do przechowywania danych wykorzystuje informacje *cookie*. Złożona aplikacja może utworzyć dane dotyczące stanu, wystarczające do przekroczenia limitów wielkości obowiązujących dla informacji *cookie*. Aplikacja z kilkoma tuzinami siatek może wygenerować informację *cookie* o wielkości, która spowoduje zawieszenie aplikacji. Zatem lepsze byłoby użycie obiektu `localStorage`, który oferuje znacznie większą wielkość, a ponadto eliminuje obciążenie wynikające z wysyłania danych do serwera w każdym żądaniu.

Okazuje się, że konfigurowanie niestandardowego obiektu dostawcy stanu jest naprawdę proste. Dostawca zaprezentowany w przykładzie 4.4 rozszerza ogólny obiekt dostawcy, a co więcej — musi zapewnić trzy metody: `set`, `clear` i `get`. Te metody po prostu odczytują dane z magazynu i zapisują je w nim. W przykładzie 4.4 zdecydowałem się na poindeksowanie danych w magazynie za pomocą raczej prostej metody polegającej na zastosowaniu łańcucha `state_` z identyfikatorem stanu zapisywanego elementu. Jest to sensowna metoda.

Przykład 4.4. Lokalny dostawca stanu biblioteki ExtJS

```
Ext.ux.LocalProvider = function() {
    Ext.ux.LocalProvider.superclass.constructor.call(this);
};
Ext.extend(Ext.ux.LocalProvider, Ext.state.Provider, {
    //*************************************************************
    set: function(name, value) {
        if (typeof value == "undefined" || value === null) {
            localStorage['state_' + name] = undefined;
            return;
        }
        else {
            localStorage['state_' + name] = this.encodeValue(value);
```


Kup książkę Poleć książkę

```
        }
    },
    //*********************************************************
    // Private
    clear: function(name) {
        localStorage['state_' + name] = undefined;
    },
    //*********************************************************
    get: function(name, defaultValue) {
        return Ext.value(this.decodeValue(localStorage['state_' + name]),
        ↪defaultValue);
    }
});
// Zdefiniowanie procedury obsługi stanu
Ext.onReady(function setupState() {
    var provider = new Ext.ux.LocalProvider();
    Ext.state.Manager.setProvider(provider);
});
```

Możliwe byłoby również umieszczenie wszystkich danych stanu w jednym dużym obiekcie i przechowywanie go w jednym kluczu magazynu. Korzyścią z tego wynikającą jest to, że w magazynie nie jest tworzona duża liczba elementów. Jednakże kod staje się bardziej złożony. Jeśli ponadto dwa okna próbują zaktualizować magazyn, jedno może negatywnie wpłynąć na zmiany dokonane przez drugie. Lokalne magazynowanie danych opisane w rozdziale nie oferuje żadnego znakomitego rozwiązania problemu dotyczącego warunków wyścigu. W miejscach, w których może to stanowić problem, prawdopodobnie lepiej będzie użyć interfejsu IndexedDB lub innego rozwiązania.

Ładowanie w trybie bez połączenia przy użyciu magazynu danych

Gdy część trwałych danych używanych w aplikacji będzie stosunkowo niezmienna, w celu zapewnienia szybszego dostępu może mieć sens ładowanie danych do lokalnego magazynu. W tym przypadku niezbędne będzie zmodyfikowanie obiektu `Ext.data.JsonStore` tak, aby jego metoda `load()` szukała danych w obszarze `localStorage` przed podjęciem próby załadowania danych z serwera. Po załadowaniu danych z obszaru `localStorage` obiekt `Ext.data.JsonStore` powinien wywołać serwer w celu sprawdzenia, czy dane zostały zmienione. W ten sposób aplikacja jest w stanie natychmiast udostępnić dane użytkownikowi, co może wiązać się z wystąpieniem krótkotrwałej niespójności. Może to wpłynąć na szybszą obsługę interfejsu przez użytkownika, a ponadto zredukowanie przepustowości wykorzystywanej przez aplikację.

Kup książkę Poleć książkę

Ponieważ w przypadku większości żądań dane nie będą modyfikowane, naprawdę sensowne jest użycie dla danych określonej postaci mechanizmu ETag. Dane są pobierane z serwera za pomocą żądania `GET` protokołu HTTP i nagłówka `If-None-Match`. Jeśli serwer stwierdzi, że dane nie zmieniły się, może wysłać odpowiedź `304 Not Modified`. Jeżeli dane zostały zmodyfikowane, serwer odeśle nowe dane, a aplikacja załaduje je zarówno do obiektu `Ext.data.JsonStore`, jak i do obiektu `sessionStorage`.

Obiekt `Ext.data.PreloadStore` (przykład 4.6) przechowuje dane w buforze sesji jako jeden duży obiekt JSON (przykład 4.5). Dodatkowo dane odsyłane przez serwer obiekt opakowuje w kopertę JSON, która umożliwia przechowywanie w niej metadanych. W tym przypadku są przechowywane dane mechanizmu ETag, a także data ich załadowania.

Przykład 4.5. Format danych trybu bez połączenia obiektu Ext.data.PreloadStore

```
{
    "etag": "25f9e794323b453885f5181f1b624d0b",
    "loadDate": "26-jan-2011",
    "data": {
        "root": [{
            "code": "us",
            "name": "Stany Zjednoczone"
        },
        {
            "code": "ca",
            "name": "Kanada"
        }]
    }
}
```

Przykład 4.6. Obiekt Ext.data.PreloadStore

```
Ext.extend(Ext.data.PreloadStore, Exta.data.JsonStore, {
  indexKey: '',
  // Domyślny klucz indeksu
  loadDefer: Time.MINUTE,
  // Określanie opóźnienia procesu ładowania danych
  listeners: {
    load: function load(store, records, options)
    {
      var etag = this.reader.etag;
      var jsonData = this.reader.jsonData;
      var data =
      {
        etag: etag,
        date: new date(),
        data: jsonData
      };
      sessionStorage[store.indexKey] = Ext.encode(data);
```


Kup książkę Poleć książkę

```
    },
beforeload: function beforeLoad(store, options)
    {
      var data = sessionStorage[store.indexKey];
      if (data === undefined || options.force)
      {
        return true; // Brak danych w buforze; ładowanie z serwera
      }
      var raw = Ext.decode(data);
      store.loadData(raw.data);
      // Odroczenie ponownego ładowania danych
      store.doConditionalLoad.defer(store.loadDefer, store, [raw.etag]);
      return false;
    }
  },
  doConditionalLoad: function doConditionalLoad(etag)
  {
    this.proxy.headers["If-None-Match"] = etag;
    this.load(
    {
      force: true
    });
  },
  forceLoad: function ()
  {
    // Przekazanie fikcyjnego identyfikatora ETag w celu wymuszenia ładowania
    this.doConditionalLoad('');
  }
});
```

Podczas tworzenia identyfikatora mechanizmu ETag należy pamiętać o użyciu odpowiedniej funkcji mieszającej. Algorytm MD5 to prawdopodobnie najlepszy wybór. Choć algorytm SHA1 też może zostać zastosowany, ponieważ generuje znacznie dłuższy łańcuch, raczej nie jest godny uwagi. W teorii możliwe jest wystąpienie kolizji dotyczącej algorytmu MD5, lecz w praktyce w przypadku kontrolowania bufora prawdopodobnie nie jest to powód do zmartwienia.

Dane w obiekcie `localStorage` mogą być zmieniane w tle. Jak już wyjaśniono, użytkownik może zmienić dane za pomocą narzędzi Developer Tools przeglądarki Chrome lub z poziomu wiersza poleceń dodatku Firebug. Zmiana danych może też po prostu wystąpić nieoczekiwanie, gdy użytkownik tę samą aplikację otworzy w dwóch przeglądarkach. A zatem ważne jest, aby magazyn prowadził nasłuch pod kątem zdarzenia aktualizacji z obiektu `localStorage`.

Kup książkę Poleć książkę

Większość operacji jest wykonywana w procedurze obsługi zdarzenia `beforeload`. Sprawdza ona, czy w magazynie danych znajduje się buforowana kopia danych. Jeśli kopia istnieje, procedura ładuje ją do magazynu. Jeżeli dane występują, procedura obsługi załaduje ponownie również dane, lecz użyje metody `Function.defer()` w celu opóźnienia procesu ładowania do momentu pomyślnego zakończenia przez system wczytywania strony internetowej. Dzięki temu z mniejszym prawdopodobieństwem proces ładowania będzie zauważalny dla użytkownika.

Metoda `store.doConditionalLoad()` kieruje wywołanie Ajax do serwera, aby załadować dane. Ponieważ metoda uwzględnia nagłówek `If-None-Match`, jeśli dane nie zmieniły się, załaduje ona bieżące dane. Metoda ta używa również opcji `force`, która sprawi, że procedura obsługi `beforeload` załaduje nowe dane i nie podejmie próby odświeżenia magazynu z wykorzystaniem buforowanej wersji obiektu `localStorage`.

Aby zwiększyć czytelność kodu, przeważnie definiuję stałe dla `SECOND`, `MINUTE` i `HOUR`.

Przechowywanie zmian w celu późniejszej synchronizacji z serwerem

W sytuacji, gdy aplikacja może działać w trybie bez połączenia lub z użyciem kiepskiego połączenia internetowego, przydatne może być zapewnienie użytkownikowi możliwości zapisu zmian bez wymogu dostępu do sieci. W tym celu zmiany w każdym rekordzie należy zapisywać w kolejce obiektu `localStorage`. Po ponownym nawiązaniu połączenia sieciowego przez przeglądarkę kolejka może być przekazana serwerowi. Pod względem działania przypomina to dziennik transakcji używany w bazie danych.

Kolejka zapisu może wyglądać podobnie jak w przykładzie 4.7. Każdy rekord w kolejce reprezentuje działanie podejmowane na serwerze. Oczywiście dokładny format będzie określany zależnie od wymaganej konkretnej aplikacji.

Przykład 4.7. Dane kolejki zapisu

```
[
  {
      "url": "/index.php",
      "params": {}
  },
  {
      "url": "/index.php",
```


Kup książkę Poleć książkę

```
        "params": {}
    },
      {
        "url": "/index.php",
        "params": {}
      }
    ]
```

Gdy przeglądarka internetowa ponownie przełączy się w tryb z połączeniem, niezbędne będzie przetworzenie elementów kolejki. W przykładzie 4.8 pierwszy element kolejki jest wysyłany do serwera. Jeśli to żądanie zakończy się powodzeniem, zostanie pobrany następny element. Proces będzie kontynuowany dla kolejnych elementów aż do momentu wysłania całej zawartości kolejki. Jeśli nawet kolejka jest długa, ten proces będzie przeprowadzany z minimalnym wpływem na działania użytkownika, ponieważ technologia Ajax przetwarza każdy element w sposób asynchroniczny. W celu zmniejszenia liczby wywołań Ajax możliwa byłaby taka modyfikacja przykładowego kodu, aby jednocześnie były wysyłane elementy kolejki w grupach liczących na przykład pięć pozycji.

Przykład 4.8. Kolejka zapisu

```
var save = function save(queue)
{
  if (queue.length > 0)
  {
    $.ajax(
    {
      url: 'save.php',
      data: queue.slice(0, 5),
      success: function (data, status, request)
      {
        save(queue.slice(5));
      }
    });
  }
};
```

Dodatki biblioteki jQuery

Jeśli niepewność wszystkich opcji magazynowania po stronie klienta nie daje spokoju, można skorzystać z innych opcji. Jak w przypadku wielu rzeczy w języku JavaScript, zły i niespójny interfejs może być zastąpiony modułem, który zapewnia znacznie lepszy interfejs. Poniżej omówiono dwa takie moduły ułatwiające pracę.

Kup książkę Poleć książkę

DSt

DSt (*http://github.com/gamache/DSt*) to prosta biblioteka, która opakowuje obiekt `localStorage`. DSt może być autonomiczną biblioteką lub działać jako dodatek biblioteki jQuery. Biblioteka DSt automatycznie przekształci dowolny złożony obiekt w strukturę JSON.

Ta biblioteka pozwala również zapisywać i przywracać stan całego formularza lub jego elementu. Aby zapisać i przywrócić element, metodzie `DSt.store()` należy przekazać element lub jego identyfikator. W celu późniejszego przywrócenia elementu należy go przekazać metodzie `DSt.recall()`.

Aby zapisać stan całego formularza, należy użyć metody `DSt.store_form()`. Metoda pobiera identyfikator lub element samego formularza. Dane mogą być przywrócone za pomocą metody `DSt.populate_form()`. Przykład 4.9 prezentuje podstawowe zastosowanie biblioteki DSt.

Przykład 4.9. Interfejs biblioteki DSt

```
$.DSt.set('key', 'value');
var value = $.DSt.get('key');

$.DSt.store('element'); // Zapisanie wartości elementu formularza
$.DSt.recall('element'); // Przywrócenie wartości elementu formularza

$.DSt.store_form('form');
$.DSt.populate_form('form');
```

jStore

Aby nie podejmować ryzyka związanego z określaniem, jakie mechanizmy magazynowania są obsługiwane przez poszczególne przeglądarki, i nie tworzyć różnego kodu dla każdego przypadku, można skorzystać z dobrego rozwiązania, czyli dodatku jStore (*http://twablet.com/docs.html?p=jstore*) dla biblioteki jQuery. Dodatek obsługuje obiekty `localStorage` i `sessionStorage`, komponenty Gears SQLite i HTML5 SQLite, a także rozwiązania Flash Storage i wbudowane w przeglądarkę Internet Explorer 7.

Dodatek jStore ma prosty interfejs, który umożliwia programiście przechowywanie par nazwa/wartość w dowolnym z różnych mechanizmów magazynowania. Dodatek zapewnia jeden zestaw interfejsów dla wszystkich mechanizmów. Z tego powodu w razie potrzeby program może dokonać degradacji z „wdziękiem” w sytuacjach, gdy mechanizm magazynowania nie jest dostępny w danej przeglądarce.

Kup książkę Poleć książkę

Dodatek jStore udostępnia listę mechanizmów obecnych w zmiennej instancji `jQuery.jStore.Availability`. Program powinien wybrać najbardziej sensowny mechanizm. W przypadku aplikacji wymagających obsługi przez wiele przeglądarek może to być pożyteczny dodatek. Więcej szczegółów można znaleźć w witrynie internetowej dodatku jStore.

Kup książkę Poleć książkę

Skorowidz

A

adres URL gniazda WWW, 135
adresowanie elementów, 55
Ajax, Asynchronous JavaScript
  and XML, 30
aktualizowanie magazynu, 90
aktualizowanie pliku manifestu, 106
aktywny plik manifestu, 105
akumulator, 40
algebra tablic, 37
algorytm
  MD5, 80
  SHA1, 80
Android, 69
aplety Java, 14
aplikacja
  ARIA, 145
  WAI-ARIA, 145
aplikacje
  internetowe, 14
  uruchamiane w przeglądarce, 14
ARIA, Accessible Rich Internet
  Applications, 145
arkusz stylów qunit, 50
asercje, 56
atak XSS, 86
atrybut itemprop, 146
atrybut role, 145
atrybuty HTML, 147

B

baza danych
  CouchDB, 85
  Gears SQLite, 72
  interfejsu IndexedDB, 88
  MongoDB, 85
  NoSQL, 85
  SQLite, 72
bezpieczeństwo danych, 86
biblioteka
  DSt, 83
  Event Machine, 140
  ExtJS, 17, 77
  jQuery, 14, 16
  jQuery Hive, 132
  PollenJS, 132
  socket.io, 135
  Symfony Yaml Library, 106
błędy w wątku roboczym, 127
bufor aplikacji, 110
buforowanie plików, 12

C

CACHE MANIFEST, 105
CDN, Content Delivery Network, 103
cena akcji spółki, 136, 138
chmura Amazon EC2, 62
ciąg Fibonacciego, 34
cookie, 71
Crockford Douglas, 16
czas ładowania strony, 49
czas wykonania operacji, 21

D

dane
  formularza, 148
  obrazu PNG, 94
  oprogramowania Gmail, 134

Kup książkę Poleć książkę

debugger
  Venkman, 16
  Weinre, 16
debugowanie
  kodu JavaScript, 16, 127
  plików manifestu, 109
deklaracja manifestu HTML, 104
Developer Tools, 74
dodatek
  Firebug, 16, 74, 95
  FireRainbow, 155
  Flash, 93
  jStore, 84
  Speed Tracer, 156
dodawanie
  magazynu danych, 87
  właściwości, 32
DOM, Document Object Model, 11
domknięcie, 21, 24
domknięcie w przycisku, 25
dostarczanie treści, 103
dostęp
  do bazy, 86
  do dysku, 20
  do funkcji, 24
  do modelu DOM, 115
  do obiektów magazynowania, 75
  do obiektu localStorage, 73
  do plików, 104
  do serwera, 133
  do systemu plików, 93, 101
  do wątków roboczych, 115
DSt, 83
dwukierunkowy kanał danych, 134
działania, 56

E

Eclipse, 155
edytowanie
  localStorage, 75
  obiektu magazynowania, 74
  sessionStorage, 75
efekty uboczne, 27
element
  canvas, 13, 149
  dokument, 55
  identyfikator, 55
  iframe, 127
  nazwa, 55
  selektor CSS, 55
  tablicy, 37
  tekst odsyłacza, 55
  wyrażenie XPath, 55
emulowanie gniazd WWW, 135

F

farma testowania, 69
Firebug, 16, 74, 95
Firefox, 111
Flash Storage, 83
format
  JSON, 30, 74, 137
  pliku manifestu, 105
  SVG, 149
  UTF-8, 139
formularz klasyczny, 147
fraktal, 120
funkcja
  Array(), 28
  buildMaster(), 123
  deepEqual(), 50
  factory(), 25
  filter(), 28
  handleButtonClick(), 47
  isDuplicate(), 92
  notDeepEqual(), 50
  notEqual(), 50
  notStrictEqual(), 50
  raises(), 50
  setTimeout(), 50
  square(), 22
  strictEqual(), 50
  tickerUpdate(), 138
  transaction.abort(), 89
  transaction.done(), 89
  WaitForElementPresent(), 64

Kup książkę Poleć książkę

funkcje
anonimowe, 22
asercji, 49
lambda, 37
mieszające, 80
uruchamiania, 128
wewnętrzne, 24
wyższego rzędu, 28
zewnętrzne, 24

G

Gears, 13
geolokacja, 116, 150
gniazdo
TCP/IP, 134
WWW, 13, 134
GWT, Google Web Toolkit, 17

H

HTTP, Hypertext Transfer Protocol, 13

I

identyfikator mechanizmu ETag, 80
instancja obiektu, 32
instrukcja
function, 22
if, 22
return, 27
instrukcje waitFor, 52
inteligentny telefon, smartphone, 14
interfejs
API, 12, 62
API FileSystem, 101
API geolokacji, 150
biblioteki DSt, 83
biblioteki jQuery, 28
BlobBuilder, 94
dla gniazd WWW, 135
FileReader, 96
gniazd WWW, 135, 138
IndexedDB, 12, 73, 85, 90
JSON, 118
localStorage, 85
MozBlobBuilder, 94
REST, 61
użytkownika, 54
WebKitBlobBuilder, 94
XHR2, 97
XMLHttpRequest, 97, 118
zdarzenia magazynowania, 74
iOS, 69

J

język
C, 13
C++, 13
CoffeeScript, 156
C#, 62
Erlang, 27, 138, 142
F#, 27
Haskell, 27, 36
HTML5, 11
Java, 13, 62, 138
JavaScript, 12, 15
Lisp, 23
Perl, 23, 62
PHP, 20, 62, 137
Python, 23, 62
Rhino, 156
Ruby, 23, 62, 138
Scala, 27
Selenese, 64
języki
funkcjonalne, 140
poleceń, 54
.NET/CLR, 138
jStore, 83

K

klasa PHPUnit_
Extensions_SeleniumTestCase, 61
klient z gniazdem, 137
klucz
główny, 91
magazynu, 78
obiektu, 86

Kup książkę Poleć książkę

kolejka zapisu, 81
komunikacja z wątkiem, 119
konfigurowanie pakietu QUnit, 47
konsorcjum W3C, 85
konstruktor MozWebSockets, 135
kontrola przepływu, 55

L

liczba
  wykonanych testów, 66
  żądań Ajax, 134
limit czasu oczekiwania, 50
lista
  domkniętych zmiennych, 26
  pozycji dozwolonych, 110

Ł

ładowanie listy plików, 106
ładowanie zestawu plików, 103
ładunek Ajax, 97
łańcuch prototypów, 31
łańcuch UTF-8, 136
łańcuchy, 31

M

magazyn
  danych, 87
  obiektów, 88
magazynowanie lokalne danych, 72
magazynowanie obiektów binarnych, 101
manifest, 103
manifest HTML5, 104
  sekcja FALLBACK, 105
  sekcja NETWORK, 105
mapy, 116
mechanizm
  ETag, 79
  localStorage, 73
  magazynowania, 85
menedżer NPM, 138
metoda
  $.decode(), 132
  $.encode(), 132
  $.get(), 132
  $.post(), 132
  $this->getText(), 64
  $this->getXpathCount(), 65
  $this->isElementPresent(), 64
  $this->isTextPresent(), 64
  alert(), 21
  BlobBuilder.getBlob(), 94
  close(), 118
  confirm(), 21
  console.info(), 123
  createObjectURL(), 95
  deleteEach(), 92
  DSt.recall(), 83
  DSt.store(), 83
  DSt.store_form(), 83
  equal(), 49
  every(), 40
  filter(), 39
  FormData.append(), 97
  Function.defer(), 81
  get, 91
  getCurrentPosition(), 150
  getEval(), 65
  getItem(), 75
  importScripts(), 118
  index(), 90
  map(), 38
  mozSlice(), 95
  ok(), 50
  onmessage(), 117
  openCursor, 91
  populate(), 32
  postMessage(), 117, 128
  prompt(), 21
  reduce(), 40
  reduceRight(), 40
  remove(), 92
  removeItem(), 75
  revokeBlobURL(), 95
  setInterval(), 36, 118
  setTimeout(), 36, 118

Kup książkę Poleć książkę

slice(), 95
socket.close(), 136
socket.onclose(), 136
socket.onopen(), 136
socket.send(), 136
some(), 40
store.doConditionalLoad(), 81
string.indexOf(), 31
string.lastIndexOf(), 31
string.match(), 31
string.replace(), 31
string.slice(), 31
string.split(), 31
then(), 91
transaction.done(), 89
webkitSlice(), 95
metody
iteracji, 37
okien, 21
uruchamiania testów, 52
mikrodane, 146
model DOM, 13
model wieloprocesorowy, 138
modyfikowanie opcji testowania, 61

N

nagłówek If-None-Match, 79, 81
nagłówki gniazda, 139
narzędzie
ClojureScript, 156
DevTools, 26
Gmail, 93
GWT, 17
JSBeautifier, 154
JSLint, 19, 153
JSMin, 154
PhoneGap, 14
Selenium Grid, 62
YSlow, 155
narzut, 135
NoSQL, 12

O

obiekt
Array.prototype, 40, 42
ArrayBuffer, 96
blob, 94, 97
DataStore, 21
do zapisu stanu, 77
Ext.data.JsonStore, 78
Ext.data.PreloadStore, 79
File, 95
FileList, 95
FileReader, 96
FormData, 97, 99
Function.prototype, 33
localStorage, 17, 72
location, 118
nasłuchujący, 98
navigator, 118
Number.prototype, 33
Object.prototype, 42
połączenia, 138
self, 118
sessionStorage, 73
String.prototype, 31
transakcji, 88
URL, 94
WebSocket, 135
window.applicationCache, 108
Worker, 117
XHR, 21, 30
XMLHttpRequest, 99
obsługa
czcionek internetowych, 151
dźwięku i wideo, 149
gniazd WWW, 135
grafiki w HTML5, 116
odchylenie standardowe, 40
okno profilu niestandardowego, 111
opakowanie, wrapper, 14
opcje odczytu danych
FileReader.readAsArrayBuffer(), 96
FileReader.readAsBinaryString(), 96
FileReader.readAsText(), 96
FileReader.readDataAsURL(), 96

Kup książkę Poleć książkę

opcje przeglądarki Firefox, 111
Opera Dragonfly, 16
operacje
  blokujące, 21
  nieblokujące, 21
  wejścia-wyjścia, 20
operator
  ==, 50
  ===, 50
  tablicowy, 99
oprogramowanie Gears, 104
otwarte żądanie HTTP, 134

P

panel administracyjny witryny, 146
panel środowiska IDE, 58
para klucz wartość, 88
parametr unlimitedStorage, 101
pasek postępu, 101
pętla for, 39
pętla zdarzeń, 113
ping, 15, 72
plik
  HTML, 61
  JAR, 68
  Java, 61
  konfiguracyjny, 104
  manifestu, 104, 105
  phpunit.xml, 61
  seleneseTest.html, 62
  archiwum, 98
pobieranie danych, 91
pole wysyłania pliku, 95
polecenia pakietu Selenium, 56
polecenie
  assertElementPresent, 57
  clickAndWait, 57
  verifyElementPresent, 57
  waitForElementPresent, 57
połączenie gniazda z serwerem, 136
położenie elementu, 54
port domyślny, 69
procedura obsługi clickHandler, 25
program seleniumrc, 61
programowanie
  funkcyjne, 27
  sterowane zdarzeniami, 20
protokół
  gniazd WWW, 139
  HTTP, 13, 135
  SSL, 135
prototyp, 31
  Function, 36
  Number, 33
przechowywanie danych, 12, 15
przechwytywacz, 33
przeciąganie plików, 93, 98
przetwarzanie
  danych mapy, 116
  elementów kolejki, 82
  w chmurze, 62
przeznaczenie witryny, 14

Q

QUnit, 47

R

rejestrowanie działań w przeglądarce, 58
rekurencja, 35
rekurencja listy, 29
relacyjna baza danych, 88
rozszerzanie
  funkcji, 33
  prototypów, 32
rozszerzenia typów podstawowych, 32
rozszerzenie
  Firebug, 16, 26
  Gears, 13
rozwijanie, currying, 36
rozwijanie obiektu, 41

S

selektor XPath, 65
selektory CSS, 150
Selenese, 54
Selenium, 50

Kup książkę Poleć książkę

serwer
  Node.js, 138, 156
  RC, 62, 68
  seleniumrc, 51
  WWW, 14
  WWW Erlang Yaws, 140
  Yaws, 138
  z gniazdami, 138
składowa statyczna $browsers, 61
skrypt pakietu Selenium, 52
słowo kluczowe lambda, 23
sprawdzanie plików, 106
standard ECMAScript 5, 118
standard XML, 149
sterownik systemu Android, 69
sterownik urządzenia iPhone, 69
Subversion, 106
suma kontrolna MD5, 106
SVG, Scalable Vector Graphics, 149
system kontroli wersji, 106
system plików, 101
szablon, 31
szybkość transmisji, 72

Ś

ścieżki do plików, 42
śledzenie stosu, 23
średnia krocząca, 38
środowisko
  Aptana, 155
  Emacs JS2 mode, 155
  IDE, 51, 58
  Java, 68
  narzędziowe Xcode, 69
  PHP, 65
  PHPUnit, 59
  piaskownicy, 14, 101
  wątków roboczych, 117
  xUnit, 52

T

tablica tools, 25
tablice, 37
technologia
  Ajax, 11, 13, 135
  Flash, 135
  Web Workers, 12, 30, 114, 127
terminal VT320, 15
test integracji, 45
test jednostkowy, 45
testowania po stronie serwera, 63
testowanie
  przeglądarek, 68
  tytułu, 63
  wątków roboczych, 127
testy
  akceptacji, 45
  integracji, 44
  jednostkowe, 44
  QUnit, 47
  Selenium, 50
transakcja jawna, 89
tryb bez połączenia, 81
tryb z połączeniem, 82
tworzenie
  bazy danych, 88
  generatora funkcji, 24
  grafiki, 149
  indeksu, 87, 90
  kodu Java, 17
  magazynu lokalnego, 72
  wywołania Ajax, 20
typ MIME, 104
typy podstawowe, 30
typy testów, 45

U

upuszczanie pliku, 98
uruchamianie
  testów, 61
  testów QUnit, 66
  pakietu Selenium, 63
usuwanie
  danych, 92
  indeksu, 87

Kup książkę Poleć książkę

uzyskiwanie
  jednego wiersza, 91
  wszystkich wierszy, 92
  zakresu wierszy, 91
użycie danych z pliku, 93

W

wartość
  this, 27
  undefined, 27, 42
  void, 96
warunek błędu, 50
wątek
  główny, 120
  roboczy, 114, 118
  roboczy Web Worker, 21
WebKit, 16, 72
Weinre, WEb INspector REmote, 16
wersja magazynu, 90
wielkość obiektu localStorage, 74
wielkość wysyłanych plików, 96
witryna
  GitHub, 50, 138
  Mozilla Developer Network, 31
  O'Reilly Answers, 19
  StackOverflow, 19
właściwość $seleneseDirectory, 62
włączanie buforu aplikacji, 110
wskaźnik Licznik, 144
wskaźnik postępu, 143, 144
wtyczka goto_sel_ide.js, 55
wyciek pamięci, 55
wyniki testów, 66
wynoszenie, hoisting, 22
wyrażenia lambda, 23
wyrażenie funkcji, 23
wyrażenie XPath, 53
wysyłanie plików, 97
wyszukiwanie sterowane głosem, 147
wyszukiwarka trasy, 116
wywołanie Ajax, 20
wywołanie zwrotne, 20
wzajemne wykluczanie, mutex, 115
wzorce plików, 105

Z

zablokowanie interfejsu, 21
zapisywanie stanu, 17
zapytania, 64
zarządzanie kolejką zadań, 126
zasięg leksykalny, 24
zbiór Mandelbrota, 123
zdarzenie
  beforeload, 81
  cached, 108
  checking, 108
  click, 57
  downloading, 108
  drop, 98
  error, 109
  magazynowania, 74
  message, 118
  mouseDown, 57
  mouseUp, 57
  noupdate, 108
  oncomplete, 97
  onprogress, 97
  progress, 108
  webkitspeechchange, 147
zmiana danych, 80
zmienna leksykalna, 24
znacznik
  <audio>, 149
  <canvas>, 116, 122
  , 47, 49
  <hr>, 145
  <img>, 145
  <input type="text">, 147
  , 145
  <meter>, 144
  <progress>, 143
  <script>, 118
  , 145
  <svg>, 116
  <video>, 116, 149
  img, 28
  speech, 147
znaczniki mikrodanych, 146
znak #, 105

Kup książkę Poleć książkę

Ż

żądanie
DELETE, 105
GET, 79, 105
POST, 105
PUT, 105
żądanie Ajax, 21
żądanie ICMP, 15

Kup książkę Poleć książkę

Kup książkę Poleć książkę

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl

GRUPA WYDAWNICZA
Helion SA

WWW

O'REILLY®

HTML5. Programowanie aplikacji

W sieci trwa właśnie rewolucja! Do władzy dochodzi język HTML5! Jego potencjał jest nieograniczony — ścisła integracja z przeglądarką internetową, wydajna obsługa grafiki czy wsparcie dla geolokalizacji to tylko niektóre z jego atutów. Korzystając z jego możliwości, stworzysz świetną grę lub przydatną aplikację internetową.

Poznaj potencjał języka HTML5 i dowiedz się, jak budować kompletne i autonomiczne aplikacje działające na urządzeniach przenośnych i konkurujące z tradycyjnymi programami. Dzięki temu praktycznemu przewodnikowi odkryjesz skuteczne metody pracy z językiem HTML5, takie jak lokalne magazynowanie danych i przetwarzanie wielowątkowe. Zaznajomisz się również z zaawansowanymi narzędziami JavaScriptu, które ułatwiają korzystanie z całej gamy elementów języka HTML5. Jeśli jesteś doświadczonym programistą JavaScriptu, umieszczone w książce przykładowe kody pokażą Ci, jak język HTML5 zamienia sieć WWW w pierwszorzędne środowisko programistyczne.

Zobacz, jak:

- testować aplikacje internetowe,
- korzystać z bazy IndexedDB,
- pracować w trybie offline,
- wykorzystać gniazda WWW.

Dołącz do zaawansowanych użytkowników HTML5!

helion.pl
księgarnia internetowa

Nr katalogowy: 8749

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion
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

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-246-4897-9

Cena 37,00 zł

Informatyka w najlepszym wydaniu

9 788324 648979