O'REILLY®

Wydanie V

PHP, MySQL i JavaScript

Wprowadzenie

Helion

Robin Nixon

Tytuł oryginału: Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5, 5th Edition

Tłumaczenie: Piotr Cieślak

ISBN: 978-83-283-5149-3

Helion SA
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/phmyj5
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

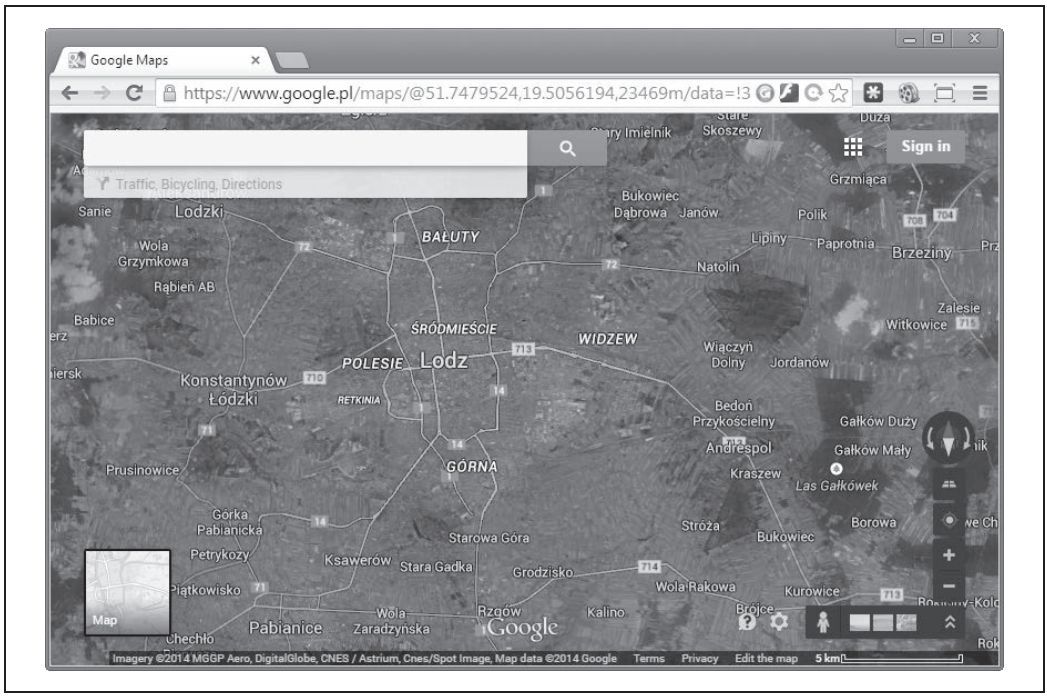
Printed in Poland.

Spis treści

Zastosowanie komunikacji asynchronicznej

Pojęcie *Ajax* po raz pierwszy zostało użyte w 2005 roku. Pochodzi ono od określenia *Asynchronous JavaScript and XML*, a bardziej przystępnie oznacza pewien zestaw wbudowanych metod języka JavaScript, służących do przesyłania danych „w tle" między przeglądarką a serwerem. Termin ten został jednak w dużej mierze zarzucony na rzecz bardziej potocznego określenia *komunikacja asynchroniczna*.

Doskonałym przykładem zastosowania tej technologii są Mapy Google (rysunek 17.1) — przeglądarka pobiera z serwera nowe części mapy dopiero wtedy, gdy są potrzebne, bez konieczności przeładowywania strony.



Rysunek 17.1. Mapy Google to doskonały przykład praktycznego zastosowania komunikacji asynchronicznej

Zastosowanie komunikacji asynchronicznej nie tylko zdecydowanie zmniejsza ilość danych przesyłanych w obu kierunkach, ale sprawia, że strony internetowe obsługuje się bardzo interaktywnie i płynnie, co upodabnia je do samowystarczalnych aplikacji. Dzięki temu uzyskuje się wygodę i responsywność interfejsu.

Czym jest komunikacja asynchroniczna?

Początki komunikacji asynchronicznej w jej dzisiejszym wydaniu można datować od premiery Internet Explorera 5, która miała miejsce w 1999 roku. Pojawił się w nim nowy obiekt ActiveX o nazwie `XMLHttpRequest`. ActiveX to technologia opracowana przez firmę Microsoft, umożliwiająca tworzenie komponentów i kontrolek rozszerzających możliwości przeglądarki. Inni producenci przeglądarek podążyli tym tropem, lecz zamiast korzystać z ActiveX, zaimplementowali podobne rozwiązania w postaci integralnej części interpretera JavaScriptu.

W pewnej uproszczonej formie mechanizmy tego rodzaju istniały jednak już wcześniej: opierały się one na użyciu na stronie ukrytych ramek, komunikujących się „w tle" z serwerem. Jednym z pierwszych zastosowań komunikacji asynchronicznej były czaty internetowe, w których technologia ta służyła do pobierania i wyświetlania nowych wiadomości bez konieczności przeładowywania strony.

Przyjrzyjmy się, jak korzystać z komunikacji asynchronicznej w JavaScripcie.

Zastosowanie obiektu XMLHttpRequest

Ze względu na różnice w implementacji obiektu `XMLHttpRequest` w różnych przeglądarkach najpierw należy napisać specjalną funkcję, która będzie gwarantowała poprawne wykonywanie kodu w każdej popularnej przeglądarce.

Aby to zrobić, należy zapoznać się z trzema sposobami tworzenia obiektu `XMLHttpRequest`:

- IE 5: `request = new ActiveXObject("Microsoft.XMLHTTP")`
- IE 6: `request = new ActiveXObject("Msxml2.XMLHTTP")`
- wszystkie pozostałe: `request = new XMLHttpRequest()`

Różnice w przeglądarce IE wynikają ze zmiany wprowadzonej przez Microsoft w Internet Explorerze 6 względem poprzedniego wydania. Wszystkie pozostałe przeglądarki używają innej metody. Kod podany w przykładzie 17.1 będzie poprawnie działał we wszystkich przeglądarkach wydanych w ciągu kilku ostatnich lat.

Przykład 17.1. Funkcja komunikacji asynchronicznej kompatybilna z wszystkimi przeglądarkami

```
<script>
  function asyncRequest()
  {
    try // Przeglądarka inna niż IE?
    {   // Tak
      var request = new XMLHttpRequest()
    }
    catch(e1)
    {
      try // IE 6+?
```

```
        {   // Tak
          request = new ActiveXObject("Msxml2.XMLHTTP")
        }
        catch(e2)
        {
          try // IE 5?
          {   // Tak
            request = new ActiveXObject("Microsoft.XMLHTTP")
          }
          catch(e3) // Brak obsługi komunikacji asynchronicznej
          {
            request = false
          }
        }
      }
      return request
    }
  </script>
```

Być może pamiętasz z rozdziału 14. wstępne informacje na temat obsługi błędów za pomocą konstrukcji `try … catch`. Przykład 17.1 doskonale ilustruje użyteczność tej metody: instrukcja `try` podejmuje próbę wykonania instrukcji dla przeglądarki innej niż IE, a jeśli próba zakończy się powodzeniem, program przeskakuje od razu do ostatniej instrukcji `return`, gdzie następuje zwrócenie nowego obiektu.

W przeciwnym razie instrukcja `catch` wyłapuje błąd i podejmowana jest kolejna próba. I ponownie: jeśli próba ta się powiedzie, zwracany jest nowy obiekt; w przeciwnym razie program przystępuje do wykonywania ostatniej spośród trzech grup instrukcji. Jeśli i ta próba zawiedzie, to znaczy, że przeglądarka nie obsługuje komunikacji asynchronicznej, a obiekt `request` otrzymuje wartość `false`; w przeciwnym razie jest zwracany w zwykły sposób. W ten sposób masz do dyspozycji uniwersalną funkcję, którą możesz z powodzeniem dodać do biblioteki przydatnych funkcji JavaScript.

Wiesz już, jak utworzyć obiekt `XMLHttpRequest`, ale co można z nim zrobić? Każdy z takich obiektów jest wyposażony w zestaw właściwości (zmiennych) oraz metod (funkcji), zebranych w tabelach 17.1 i 17.2.

Tabela 17.1. Właściwości obiektu XMLHttpRequest

Właściwość	Opis
`onreadystatechange`	Określa zdarzenie wywoływane za każdym razem, gdy właściwość `readyState` obiektu ulegnie zmianie.
`readyState`	Właściwość przyjmująca wartości całkowite, informująca o bieżącym stanie żądania. Może mieć następującą wartość: 0 – niezainicjalizowane, 1 – przesyłanie, 2 – przesłane, 3 – przetwarzanie, 4 – zakończone.
`responseText`	Dane w postaci tekstowej zwrócone przez serwer.
`responseXML`	Dane w postaci XML zwrócone przez serwer.
`status`	Kod statusu HTTP zwrócony przez serwer.
`statusText`	Tekst statusu HTTP zwrócony przez serwer.

Tabela 17.2. Metody obiektu XMLHttpRequest

Metoda	Opis
`abort()`	Przerywa bieżące żądanie.
`getAllResponseHeaders()`	Zwraca wszystkie nagłówki w postaci łańcucha znaków.
`getResponseHeader(parametr)`	Zwraca wartość `param` w postaci łańcucha znaków.
`open('metoda', 'url', 'asynch')`	Określa metodę przesyłania danych HTTP (`GET` albo `POST`), docelowy adres URL oraz czy żądanie ma być obsłużone asynchronicznie (`true` albo `false`).
`send(dane)`	Wysyła `dane` do docelowego serwera przy użyciu podanej metody HTTP.
`setRequestHeader('parametr', 'wartość')`	Określa nagłówek w postaci pary parametr/wartość.

Te właściwości i metody dają kontrolę nad rodzajem danych wysyłanych na serwer i odbieranych z niego, a także umożliwiają określenie sposobu ich wysyłania i odbierania. Można na przykład określić, czy dane mają zostać przesłane w postaci czystego tekstu (który może zawierać kod HTML i inne znaczniki), czy w formie XML, a także wybrać metodę `POST` lub `GET` wysyłania danych na serwer.

Przyjrzyjmy się najpierw metodzie `POST` na podstawie dwóch bardzo prostych dokumentów: pierwszy zawiera kod HTML i JavaScript, a drugi — program PHP, który komunikuje się asynchronicznie z pierwszym. Wystarczy kilka linii kodu JavaScript, by pobrać z zewnętrznego serwera dokument, który następnie jest zwracany do przeglądarki za pośrednictwem Twojego serwera i umieszczany w wybranej części bieżącego dokumentu.

Twój pierwszy program asynchroniczny

Wprowadź kod podany w przykładzie 17.2 i zapisz go pod nazwą *urlpost.html*, ale na razie nie otwieraj go w przeglądarce.

Przykład 17.2. Dokument urlpost.html

```
<!DOCTYPE html>
<html> <!-- urlpost.html -->
  <head>
    <title>Przykład komunikacji asynchronicznej</title>
  </head>
  <body style='text-align:center'>
    <h1>Wczytywanie strony do elementu DIV</h1>
    <div id='info'>To zdanie zostanie zastąpione</div>

    <script>
      params  = "url=news.com"
      request = new asyncRequest()

      request.open("POST", "urlpost.php", true)
      request.setRequestHeader("Content-type",
        "application/x-www-form-urlencoded")
      request.setRequestHeader("Content-length", params.length)
      request.setRequestHeader("Connection", "close")

      request.onreadystatechange = function()
```

```
    {
      if (this.readyState == 4)
      {
        if (this.status == 200)
        {
          if (this.responseText != null)
          {
            document.getElementById('info').innerHTML =
              this.responseText
          }
          else alert("Błąd komunikacji: Nie otrzymano danych")
        }
        else alert( "Błąd komunikacji: " + this.statusText)
      }
    }

    request.send(params)

    function asyncRequest()
    {
      try
      {
        var request = new XMLHttpRequest()
      }
      catch(e1)
      {
        try
        {
          request = new ActiveXObject("Msxml2.XMLHTTP")
        }
        catch(e2)
        {
          try
          {
            request = new ActiveXObject("Microsoft.XMLHTTP")
          }
          catch(e3)
          {
            request = false
          }
        }
      }
      return request
    }
  </script>
 </body>
</html>
```

Przyjrzyjmy się temu przykładowi i jego działaniu. Kilka pierwszych linii odpowiada za zainicjowanie dokumentu HTML i wyświetlenie nagłówka. W następnej linii jest tworzony element `<div>` o identyfikatorze `info`, który zawiera napis `To zdanie zostanie zastąpione`. Później pojawi się tutaj tekst zwrócony przez wywołanie asynchroniczne.

Następnych sześć linii jest niezbędne do zrealizowania żądania HTTP `POST`. Najpierw zmiennej `params` zostaje przypisana para w postaci *`parametr=wartość`*, która zostanie przesłana na serwer. Następnie jest tworzony nowy obiekt żądania. Potem za pomocą metody `open` zostaje określony sposób przesłania żądania (`POST`), adres (*urlpost.php*) oraz tryb (asynchroniczny). Trzy ostatnie linie w tej sekcji definiują nagłówki, które poinformują docelowy serwer o żądaniu `POST`.

Właściwość readyState

Teraz możemy zająć się detalami obsługi wywołań asynchronicznych, która opiera się na właściwości `readyState`. Wywołania te umożliwiają przeglądarce ciągłą interakcję z użytkownikiem i zmianę wyświetlanych danych, podczas gdy program za pośrednictwem właściwości `onreadystatechange` odwołuje się do funkcji wykonywanej przy każdej zmianie właściwości `readyState`. W tym przypadku została użyta funkcja anonimowa, umieszczona bezpośrednio w wierszu z żądaniem — można jednak w tym celu użyć osobnej funkcji o zdefiniowanej nazwie. Tego rodzaju funkcje noszą niekiedy nazwę *wywołań zwrotnych* (ang. *callback*), gdyż program zwraca się do nich za każdym razem, gdy zmieni się właściwość `readyState`.

Składnia wywołania zwrotnego przy założeniu użycia funkcji anonimowej w jednym wierszu wygląda następująco:

```
request.onreadystatechange = function()
{
 if (this.readyState == 4)
 {
   // tu coś się dzieje
 }
}
```

Gdybyś chciał użyć osobnej funkcji o zdefiniowanej nazwie, konstrukcja będzie wyglądała tylko trochę inaczej:

```
request.onreadystatechange = asyncCallback
function asyncCallback()
{
 if (this.readyState == 4)
 {
   // tu coś się dzieje
 }
}
```

Na podstawie tabeli 17.1 wiesz, że właściwość `readyState` może przyjmować pięć wartości. Ale nas interesuje tylko jedna: wartość `4`, która oznacza zakończenie żądania. Jak widać, wywołanie naszej nowej funkcji nie będzie miało żadnych efektów, dopóki właściwość `readyState` nie przyjmie wartości `4`. Gdy wartość ta zostanie wykryta, sprawdzany jest `status` żądania, aby się upewnić, że ma on wartość `200` — taka wartość oznacza bowiem, że żądanie zostało zakończone powodzeniem. Jeśli wartość jest inna niż `200`, na ekranie pojawia się komunikat błędu zawarty we właściwości `statusText`.

Zauważ, że zamiast nazwy używanego obiektu, czyli `request` (np. `request.readyState` lub `request.status`), w odwołaniach do właściwości obiektów pojawia się słowo `this` (np. `this.readyState`, `this.status` itd.). Chodzi o to, by móc łatwo przenieść kod do innego programu, który w tej postaci będzie działał z dowolnie nazwanym obiektem: słowo kluczowe `this` zawsze odnosi się bowiem do bieżącego obiektu.

Po upewnieniu się, że właściwość `readyState` ma wartość `4`, a `status` wynosi `200`, należy sprawdzić właściwość `responseText`, aby się przekonać, czy zawiera ona jakąś wartość. Jeśli nie, na ekranie pojawi się okno z ostrzeżeniem. W przeciwnym razie treść elementu `<div>` zostanie zastąpiona wartością właściwości `responseText`:

```
document.getElementById('info').innerHTML = this.responseText
```

W tej linii kodu metoda `getElementByID` odwołuje się do elementu `info`, a wartość zwrócona przez to odwołanie jest przypisywana do właściwości `innerHTML` tego elementu. Efekt jest taki, że zmienia się tylko ten element strony internetowej, a wszystko inne pozostaje bez zmian.

Po wszystkich tych przygotowaniach żądanie asynchroniczne jest wreszcie wysyłane na serwer przy użyciu następującej instrukcji, która przekazuje parametry zdefiniowane uprzednio w zmiennej `params`:

```
request.send(params)
```

Omówiona procedura jest uaktywniana za każdym razem, gdy wartość właściwości `readyState` ulega zmianie.

Pozostała część kodu to funkcja `asyncRequest` z przykładu 17.1, a także znaczniki kończące skrypt oraz dokument HTML.

Proces komunikacji asynchronicznej po stronie serwera

Przejdźmy teraz do drugiej części procedury, realizowanej za pomocą PHP — jej kod został podany w przykładzie 17.3. Przepisz go i zapisz pod nazwą *urlpost.php*.

Przykład 17.3. Dokument urlpost.php

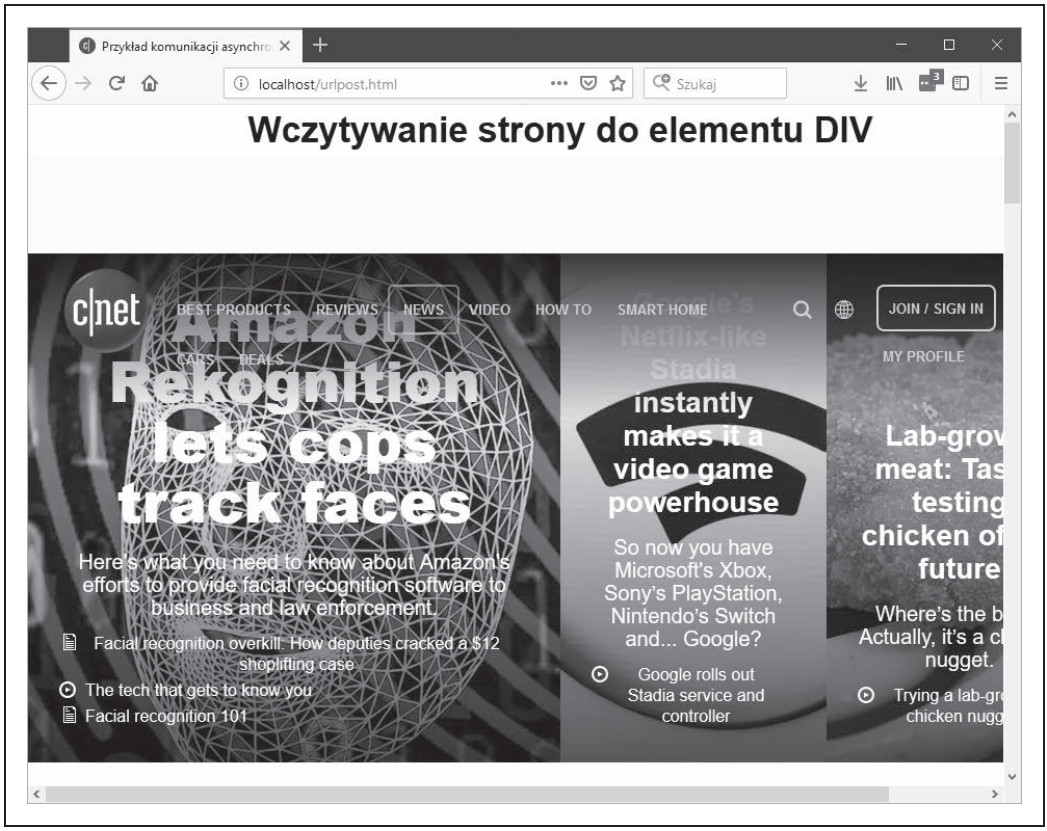
```
<?php // urlpost.php
  if (isset($_POST['url']))
  {
    echo file_get_contents('http://' . SanitizeString($_POST['url']));
  }

  function SanitizeString($var)
  {
    $var = strip_tags($var);
    $var = htmlentities($var);
    return stripslashes($var);
  }
?>
```

Jak widać, kod jest krótki, prosty i — tak jak to powinno być w przypadku wszystkich przesyłanych danych — robi dobry użytek z bardzo ważnej funkcji `SanitizeString`. W tym przypadku pominięcie procedury oczyszczania danych mogłoby doprowadzić do wstawienia przez użytkownika kodu JavaScript i przejęcia kontroli nad Twoim programem.

Program korzysta z funkcji PHP o nazwie `file_get_contents` do pobrania strony internetowej znajdującej się pod adresem URL przekazanym za pośrednictwem zmiennej `$_POST['url']`. Funkcja `file_get_contents` jest o tyle uniwersalna, że umożliwia załadowanie całej zawartości pliku albo strony internetowej z lokalnego lub zdalnego serwera; uwzględnia ona nawet przekierowania (na przykład w przypadku przeniesionych stron).

Po zapisaniu pliku z podanym kodem możesz otworzyć dokument *urlpost.html* w przeglądarce i po kilku chwilach powinieneś zobaczyć stronę internetową serwisu *news.com* wczytaną do elementu `<div>`, utworzonego specjalnie w tym celu. Cały proces będzie trwał trochę dłużej od bezpośredniego otwarcia strony WWW, gdyż dokument zostanie przesłany dwukrotnie: raz na serwer i po raz kolejny z serwera do przeglądarki. Efekt powinien wyglądać podobnie jak na rysunku 17.2.



*Rysunek 17.2. Strona główna serwisu news.com wczytana do elementu *

W ten sposób udało się nam nie tylko wykonać żądanie asynchroniczne i zwrócić odpowiedź do skryptu JavaScript, ale przy okazji wykorzystać możliwości PHP do połączenia zupełnie niezależnych stron WWW. Tak się bowiem składa, że jeśli spróbowalibyśmy bezpośrednio pobrać tę stronę internetową za pomocą żądania asynchronicznego (z pominięciem skryptu PHP na serwerze), nie udałoby się nam to ze względu na istnienie zabezpieczeń chroniących przed wykonywaniem takich żądań między różnymi domenami. Ten prosty przykład stanowi więc zarazem rozwiązanie praktycznego problemu.

Zastosowanie metody GET zamiast POST

Tak jak w przypadku przesyłania dowolnych danych za pośrednictwem formularza, istnieje możliwość przekazania danych w trybie `GET`, co zarazem pozwala zmniejszyć objętość kodu o kilka linii. Ma to jednak pewną wadę: niektóre przeglądarki mogą przechowywać żądania `GET` w pamięci cache, co w przypadku żądań `POST` nigdy nie ma miejsca. Żądania nie powinny być jednak przechowywane w pamięci podręcznej, bo przeglądarka po prostu ponownie wyświetli to, co otrzymała poprzednio, zamiast zwrócić się do serwera o nowe dane. Pewnym obejściem tego problemu jest dodanie do każdego żądania losowego parametru, dzięki czemu docelowy adres URL jest za każdym razem inny.

Przykład 17.4 pokazuje, w jaki sposób uzyskać efekt analogiczny jak w przypadku przykładu 17.2, ale za pomocą żądań typu GET zamiast POST.

Przykład 17.4. Dokument urlget.html

```
<!DOCTYPE html>
<html> <!-- urlget.html -->
  <head>
    <title>Przykład komunikacji asynchronicznej</title>
  </head>
  <body style='text-align:center'>
    <h1>Wczytywanie strony do elementu DIV</h1>
    <div id='info'>To zdanie zostanie zastąpione</div>

    <script>
      nocache = "&nocache=" + Math.random() * 1000000
      request = new asyncRequest()
      request.open("GET", "urlget.php?url=news.com" + nocache, true)

      request.onreadystatechange = function()
      {
        if (this.readyState == 4)
        {
          if (this.status == 200)
          {
            if (this.responseText != null)
            {
              document.getElementById('info').innerHTML =
                this.responseText
            }
            else alert("Błąd komunikacji: Nie otrzymano danych")
          }
          else alert( "Błąd komunikacji: " + this.statusText)
        }
      }

      request.send(null)

      function asyncRequest()
      {
        try
        {
          var request = new XMLHttpRequest()
        }
        catch(e1)
        {
          try
          {
            request = new ActiveXObject("Msxml2.XMLHTTP")
          }
          catch(e2)
          {
            try
            {
              request = new ActiveXObject("Microsoft.XMLHTTP")
            }
            catch(e3)
            {
```

```
            request = false
          }
        }
      }
      return request
    }
  </script>
 </body>
</html>
```

Najważniejsza różnica między dwoma dokumentami została wyróżniona pogrubieniem. Można ją opisać następująco:

- W przypadku żądania typu `GET` nie trzeba przesyłać nagłówków.
- W żądaniach typu `GET` używamy metody `open`. Przekazujemy jej URL zawierający symbol `?`, po którym następuje para parametr-wartość w postaci (tutaj) `url=news.com`.
- Drugą parę parametr/wartość dołączamy za pomocą symbolu `&`. Parametr nosi nazwę `nocache`, a jego wartość jest losowana z zakresu od 0 do 1000000. Dzięki temu każdy URL będzie inny, a kolejne żądania nie będą pobierane z pamięci podręcznej przeglądarki.
- W odwołaniu do metody `send` przekazujemy teraz tylko argument `null`, gdyż w przypadku metody `GET` nie są przekazywane żadne parametry. Warto pamiętać, że całkowite zrezygnowanie z przekazywania argumentów nie wchodzi w grę, gdyż wywołałoby błąd.

Aby nowy dokument był kompletny, trzeba jeszcze zmodyfikować program w PHP tak, by obsługiwał żądanie `GET`. Przykład 17.5 należy zapisać w pliku o nazwie *urlget.php*.

Przykład 17.5. Dokument urlget.php

```
<?php // urlget.php
  if (isset($_GET['url']))
  {
    echo file_get_contents("http://".sanitizeString($_GET['url']));
  }

  function sanitizeString($var)
  {
    $var = strip_tags($var);
    $var = htmlentities($var);
    return stripslashes($var);
  }
?>
```

Cała różnica pomiędzy powyższym kodem a przykładem 17.3 sprowadza się do tego, że odwołania do tablicy `$_POST` zostały zastąpione odwołaniami do tablicy `$_GET`. Efekt otwarcia dokumentu *urlget.html* w przeglądarce jest taki sam jak *urlpost.html*.

Przesyłanie żądań XML

Choć obiekty, które tworzyliśmy do tej pory, należą do klasy o nazwie `XMLHttpRequest`, jak dotąd w ogóle nie posługiwaliśmy się XML. Przekonałeś się już, że za pośrednictwem żądań asynchronicznych bez trudu da się pobrać cały dokument HTML, ale równie dobrze moglibyśmy zażądać strony czysto tekstowej, łańcucha znaków, liczby, a nawet danych z arkusza kalkulacyjnego.

Zmodyfikujmy zatem poprzedni przykładowy dokument HTML i program PHP w taki sposób, by pobrać dane XML. Aby to zrobić, najpierw przyjrzyjmy się programowi *xmlget.php* pokazanemu w przykładzie 17.6.

Przykład 17.6. Dokument xmlget.php

```
<?php // xmlget.php
  if (isset($_GET['url']))
  {
    header('Content-Type: text/xml');
    echo file_get_contents("http://".sanitizeString($_GET['url']));
  }

  function sanitizeString($var)
  {
    $var = strip_tags($var);
    $var = htmlentities($var);
    return stripslashes($var);
  }
?>
```

Jak widać, program został bardzo nieznacznie zmodyfikowany (zmiana została wyróżniona pogrubieniem) w celu przesłania poprawnego nagłówka XML przed odwołaniem się do docelowego dokumentu. W programie nie zastosowano żadnych mechanizmów sprawdzających — przyjąłem założenie, że żądanie będzie rzeczywiście odwoływało się do dokumentu XML.

Przejdźmy teraz do kodu HTML w pliku *xmlget.html* w przykładzie 17.7.

Przykład 17.7. Dokument xmlget.html

```
<!DOCTYPE html>
<html> <!-- xmlget.html -->
  <head>
    <meta charset="utf-8">
    <title>Przykład komunikacji asynchronicznej</title>
  </head>
  <body>
    <h1>Wczytywanie danych XML do elementu DIV</h1>
    <div id='info'>To zdanie zostanie zastąpione</div>

    <script>
      nocache = "&nocache=" + Math.random() * 1000000
      url     = "rss.news.yahoo.com/rss/topstories"
      out     = "";

      request = new asyncRequest()
      request.open("GET", "xmlget.php?url=" + url + nocache, true)

      request.onreadystatechange = function()
      {
        if (this.readyState == 4)
        {
          if (this.status == 200)
          {
            if (this.responseText != null)
            {
              titles = this.responseXML.getElementsByTagName('title')
```

```
            for (j = 0 ; j < titles.length ; ++j)
            {
              out += titles[j].childNodes[0].nodeValue + '<br>'
            }
            document.getElementById('info').innerHTML = out
          }
          else alert("Błąd komunikacji: Nie otrzymano danych")
        }
        else alert( "Błąd komunikacji: " + this.statusText)
      }
    }

    request.send(null)

    function asyncRequest()
    {
      try
      {
        var request = new XMLHttpRequest()
      }
      catch(e1)
      {
        try
        {
          request = new ActiveXObject("Msxml2.XMLHTTP")
        }
        catch(e2)
        {
          try
          {
            request = new ActiveXObject("Microsoft.XMLHTTP")
          }
          catch(e3)
          {
            request = false
          }
        }
      }
      return request
    }
  </script>
 </body>
</html>
```

Po raz kolejny zmiany zostały wyróżnione pogrubieniem, co pozwala łatwo zauważyć, że kod jest bardzo podobny do poprzednich wersji — z tym że docelowy URL *rss.news.yahoo.com/rss/topstories* odwołuje się do dokumentu XML, a konkretnie do kanału RSS z najpopularniejszymi wiadomościami z serwisu Yahoo!.

Inna istotna różnica polega na zastosowaniu właściwości `responseXML`, która zastąpiła właściwość `responseText`. Gdy serwer zwraca dane XML, zostają one zapisane właśnie w `responseXML`.

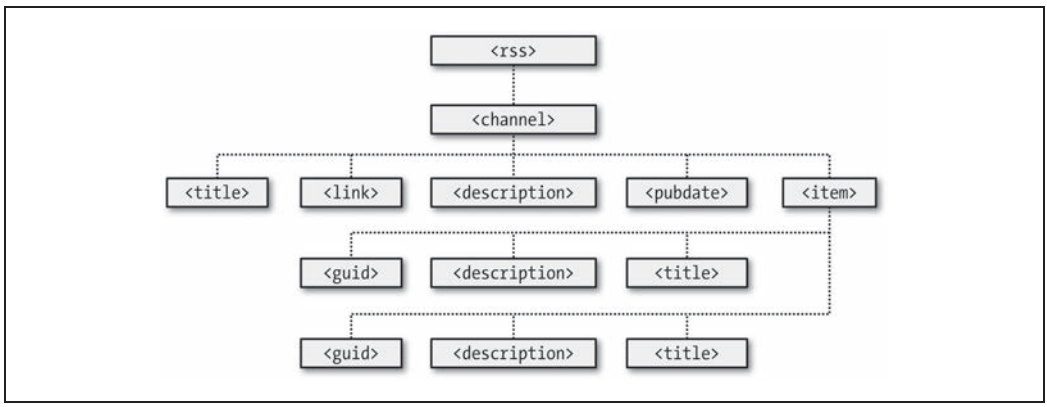
Właściwość `responseXML` nie zawiera jednak zwykłego łańcucha tekstowego XML: tak naprawdę jest to obiekt z dokumentem XML, który można analizować i przetwarzać przy użyciu metod i właściwości typowych dla drzewa DOM. To oznacza, że do takiego obiektu można się odwołać na przykład przy użyciu metody JavaScript `getElementsByTagName`.

Kilka słów o XML

Dokumenty XML, o jakich tutaj mowa, na ogół mają postać danych RSS, jak w przykładzie 17.8. Piękno XML polega jednak na tym, że strukturę tego rodzaju można zapisać w ramach drzewa DOM (rysunek 17.3), co umożliwia jej szybkie i wygodne przeszukiwanie.

Przykład 17.8. Przykładowy dokument XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
  <channel>
    <title>RSS Feed</title>
    <link>http://stronainternetowa.com</link>
    <description>Kanał RSS serwisu stronainternetowa.com</description>
    <pubDate>Mon, 11 May 2020 00:00:00 GMT</pubDate>
    <item>
      <title>Nagłówek</title>
      <guid>http://stronainternetowa.com/naglowek</guid>
      <description>To jest nagłówek</description>
    </item>
    <item>
      <title>Nagłówek 2</title>
      <guid>http://website.com/naglowek2</guid>
      <description>Drugi nagłówek</description>
    </item>
  </channel>
</rss>
```



Rysunek 17.3. Drzewo DOM z przykładu 17.8

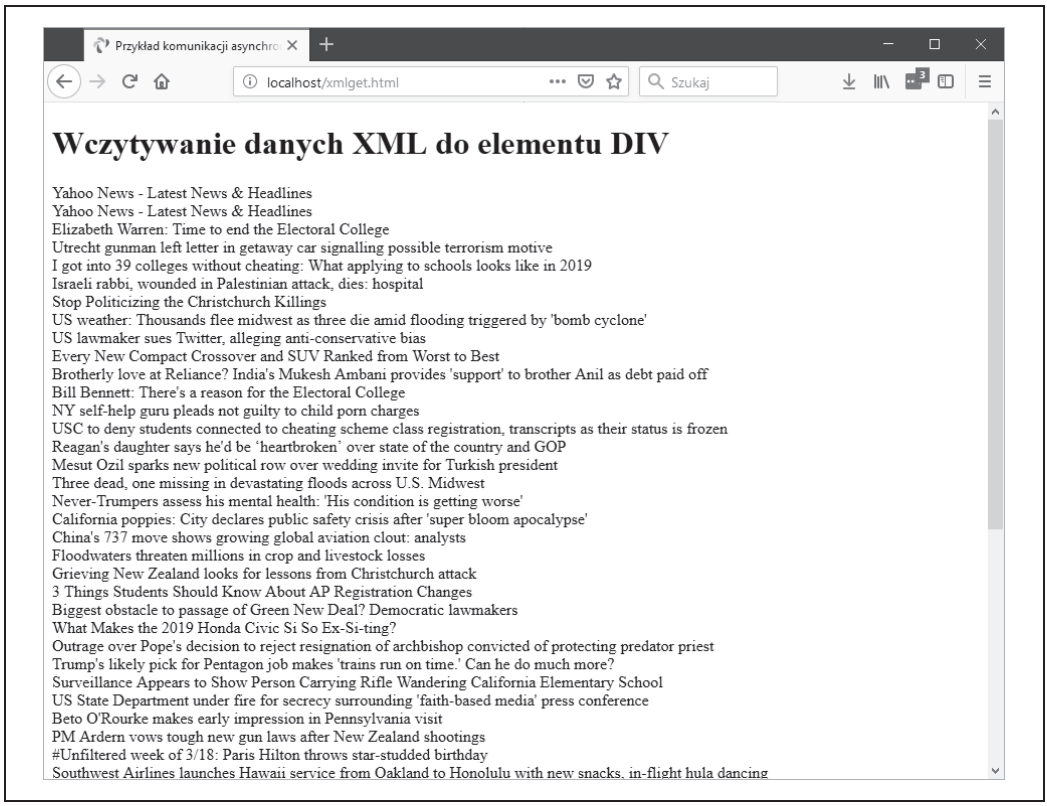
Za pomocą metody `getElementsByTagName` można szybko wyszukać wartości powiązane z różnymi znacznikami, bez czasochłonnego przeszukiwania całego tekstu. Właśnie taki jest cel użycia poniższej instrukcji w kodzie przykładu 17.7:

```
titles = this.responseXML.getElementsByTagName('title')
```

Ta jedna instrukcja wystarczy, by wszystkie wartości elementów `<title>` — czyli tytułów wiadomości — zostały umieszczone w tablicy `titles`. Stąd zaś już tylko krok, by wyodrębnić je przy użyciu następującej składni (gdzie zmienna `j` zawiera wartość całkowitą odpowiadającą pozycji tytułu, do którego się odwołujemy):

```
titles[j].childNodes[0].nodeValue
```

Wszystkie tytuły są następnie dołączane do zmiennej tekstowej `out`, a po ich przetworzeniu rezultat jest umieszczany w elemencie `<div>` na początku dokumentu. Po otwarciu pliku *xmlget.html* w przeglądarce powinieneś otrzymać efekt podobny jak na rysunku 17.4.



Rysunek 17.4. Asynchroniczne pobieranie nagłówków wiadomości w postaci XML z serwisu Yahoo!

Podsumowując, każdy składnik taki jak `title` jest tzw. węzłem (ang. *node*), to zaś oznacza, że tekst tytułu jest uważany za węzeł. Odwołując się do węzła potomnego, trzeba zażądać go w postaci tekstowej — i temu służy `.nodeValue`. Ponadto przy żądaniu danych XML — tak jak w przypadku formularzy — można użyć metody `POST` albo `GET`; wybór nie ma większego wpływu na rezultat.

Po co używać XML?

Zastanawiasz się być może, po co używać XML w celach innych niż pobieranie dokumentów XML takich jak kanały informacyjne RSS. No cóż, rzeczywiście nie ma takiej konieczności, ale gdyby zależało Ci na tym, by dane przesyłane do aplikacji Ajax miały konkretną strukturę, to jest to znacznie wygodniejsze niż przetwarzanie zwykłego tekstu przy użyciu procedur w JavaScripcie.

W takich przypadkach lepiej utworzyć dokument XML i przekazać go z powrotem do funkcji wywołującej, która automatycznie umieści go w drzewie DOM, równie łatwym do przetwarzania jak drzewo DOM dokumentu HTML, który poznałeś już wcześniej.

Obecnie programiści częściej stosują do wymiany danych JavaScript Object Notation (*http://json.org*), czyli JSON, prosty podzbiór JavaScriptu.

Zastosowanie bibliotek komunikacji asynchronicznej

Wiesz już, w jaki sposób od podstaw programować procedury komunikacji asynchronicznej, zachęcam Cię więc do zapoznania się z darmowymi bibliotekami (ang. *frameworks*), które zdecydowanie ułatwiają pracę z tą technologią i oferują wiele zaawansowanych funkcji. Szczególnie chciałbym Ci polecić przyjrzenie się bibliotece jQuery, która jest chyba najpopularniejszym rozwiązaniem tego typu — będziesz zresztą mógł o niej przeczytać w rozdziale 21. Tymczasem jednak w kolejnym rozdziale przyjrzymy się modyfikowaniu wyglądu stron WWW przy użyciu technologii CSS.

Pytania

1. Dlaczego należy napisać specjalną funkcję do tworzenia nowych obiektów `XMLHttpRequest`?
2. Do czego służy konstrukcja `try … catch`?
3. Ile właściwości i metod ma obiekt `XMLHttpRequest`?
4. Jak sprawdzić, czy przetwarzanie żądania asynchronicznego zostało zakończone?
5. Jak sprawdzić, czy przetwarzanie żądania asynchronicznego zakończyło się powodzeniem?
6. Jaka właściwość obiektu `XMLHttpRequest` zwraca odpowiedź w postaci tekstowej na żądanie asynchroniczne?
7. Jaka właściwość obiektu `XMLHttpRequest` zwraca odpowiedź w postaci XML na żądanie asynchroniczne?
8. Jak odwołać się do funkcji zwrotnej obsługującej odpowiedzi na żądania asynchroniczne?
9. Jaka metoda obiektu `XMLHttpRequest` służy do inicjalizowania żądania asynchronicznego?
10. Na czym polegają główne różnice między żądaniami asynchronicznymi typu `GET` i `POST`?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 17.”.

Skorowidz

C

D

E

F

G

H

I

J

K

L

L

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

Z

O'REILLY®

PHP, MySQL i JavaScript: klasyczne narzędzia dla nowoczesnych rozwiązań!

Trio: PHP, MySQL i JavaScript jest znane jako najwygodniejszy zestaw narzędzi do tworzenia dynamicznych stron internetowych, które do działania wymagają bazy danych. Mimo upływu lat i dynamicznego rozwoju konkurencyjnych technologii twórcy aplikacji WWW wciąż cenią PHP, MySQL i JavaScript za otwarte źródła, brak opłat za korzystanie, elastyczność i łatwość w nauce. Każdy ambitny programista posługujący się systemem Unix czy Linux z serwerem Apache powinien zapoznać się z tymi narzędziami. W połączeniu z takimi technologiami jak jQuery, CSS i HTML5 pozwalają budować serwisy porównywalne z gigantami w rodzaju Facebooka, Twittera czy Gmaila.

Ta książka jest kolejnym, uzupełnionym i zaktualizowanym wydaniem znakomitego wprowadzenia do projektowania dynamicznych stron internetowych. Oprócz przystępnego omówienia technik tworzenia responsywnych stron WWW znalazły się tu solidne podstawy PHP, MySQL, JavaScriptu, CSS i HTML5; opisano też możliwości bibliotek jQuery i jQuery Mobile. Pokazano, jak połączyć możliwości tych technologii, oraz opisano zalecane metody programowania. Co istotne, w książce zawarto wskazówki dotyczące optymalizacji stron WWW pod kątem urządzeń mobilnych. Dzięki licznym przykładom uzyskaną wiedzę można na bieżąco testować w praktyce.

W tej książce między innymi

- baza danych i zapytania MySQL
- podstawy tworzenia dynamicznych stron PHP
- mechanizm sesji i zachowanie zasad bezpieczeństwa
- biblioteki jQuery i jQuery Mobile oraz żądania Ajax
- CSS2 i CSS3 oraz nowe funkcje HTML5: geolokacja, obsługa dźwięku i filmów, Canvas

Robin Nixon – od ponad 40 lat tworzy oprogramowanie, strony internetowe i aplikacje. Jest autorem przeszło 500 artykułów, ponad 30 książek oraz internetowych kursów wideo. Interesują go psychologia, motywacja, sztuczna inteligencja i różne gatunki muzyki; z pasją oddaje się studiom nad filozofią i kulturą.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-5149-3

INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 119,00 zł