Ajax. Od podstaw

Autor: Chris Ullman, Lucinda Dykes
Tłumaczenie: Anna Trojan
ISBN: 978-83-246-1287-1
Tytuł oryginału: Beginning Ajax (Programmer to Programmer)
Format: B5, stron: około 540
oprawa twarda

Wprowadzenie do technologii Ajax dla webmasterów

- Jak tworzyć interaktywne aplikacje webowe w technologii Ajax?
- Jak pisać i wykorzystywać wzorce projektowe w Ajax?
- Dlaczego warto skorzystać z możliwości Ajaksa?

Choć technologia znana jest od końca ubiegłego wieku, uznanie zdobyła dopiero parę lat temu, kiedy Google zaczęło używać jej na większą skalę w swoich projektach. Wtedy webmasterzy odkryli, jaka siła tkwi w połączonych technologiach, kryjących się pod nazwą Ajax. Dołącz do nich i wzbogać swoje strony WWW, co sprawi, że staną się interaktywne bardziej niż kiedykolwiek.

„Ajax. Od podstaw" to książka dla twórców witryn internetowych, którzy chcą poszerzyć swoje umiejętności o niesamowitą technologię Ajax. Chris Ullmann i Lucinda Dykes, autorzy i współautorzy wielu książek o technikach programowania, przekazują tu swoje wieloletnie doświadczenie. Ułatwiają czytelnikowi opanowanie języka, przechodząc od podstawowych do coraz bardziej złożonych przypadków opartych na rzeczywistych przykładach. Narracyjny styl książki sprawia, że jest przyjazna czytelnikowi, a usystematyzowany układ czyni ją przejrzystą.

- Model i składnia aplikacji opartej na technologii Ajax
- Komunikacja z serwerami
- Synchroniczne i asynchroniczne przesyłanie danych
- Usługi sieciowe
- API oraz aplikacje mushup
- Praca z XML, XSLT oraz XPath
- Integracja z formatem wymiany danych JSON
- Praca z serwerem MySQL
- Usuwanie oraz obsługa błędów
- Wzorce projektowe
- Wykorzystanie danych zewnętrznych
- Platformy oraz biblioteki dostępne dla technologii Ajax
- Przewodnik po języku JavaScript

Poznaj nowe sposoby programowania webowego i wzbogać swoje witryny WWW o technologię Ajax!

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

3

Ajax oraz technologie po stronie serwera

Po lekturze wcześniejszych rozdziałów można odnieść wrażenie, że Ajax to tylko to, co dzieje się po stronie klienta. W początkowej definicji Ajaksa określenie, że współpracuje on „z danymi wejściowymi ze strony serwera" było istotnym ograniczeniem. Tak naprawdę wielu programistów Ajaksa uważało, że aplikacja używająca `XMLHttpRequest` do uaktualnienia strony bez danych wejściowych z serwera nie jest prawdziwą aplikacją ajaksową. W omówionych dotychczas kwestiach unikano problemu drugiej strony komunikacji klient-serwer — strony serwera. W rzeczywistości cała ta sytuacja przypomina rozmowę telefoniczną, w której mówi się, jednak nie jest się w stanie słyszeć, co odpowiada rozmówca. W dwóch pierwszych rozdziałach niniejszej książki w ogóle nie używano żadnego kodu po stronie serwera. Musi się to zatem zmienić.

Jednym z początkowych założeń tej książki było pozostanie niezależnym i niejednoznacznym w kwestii wyboru technologii po stronie serwera. Tak naprawdę w niniejszym rozdziale założono, że Czytelnik dokonał już wyboru, dlatego pisanie o tym, jakie technologie preferują autorzy, mogłoby tylko prowadzić do antagonizmów. Jedną z istotniejszych zalet Ajaksa jest to, że w aplikacjach opartych na tej technologii można z łatwością użyć dowolnej technologii po stronie serwera. Zostanie to zademonstrowane w niniejszym rozdziale.

Sposób, w jaki Ajax (czyli obiekt `XMLHttpRequest`) komunikuje się z serwerem, jest w każdej technologii w zasadzie taki sam. Różnice polegają jednak na sposobie obsługi danych, które dotrą do serwera. Oczywiste jest, że niemożliwe byłoby omówienie wszystkich technologii po stronie serwera w jednym rozdziale, dlatego autorzy książki wybrali trzy z nich, uważane za najbardziej popularne w kontekście Ajaksa: ASP.NET, PHP oraz Javę. Jeśli Czytelnik pracuje w innej technologii, może się przyjrzeć zaprezentowanym tutaj wzorcom i pomyśleć, w jaki sposób można je zastosować w innych warunkach.

W omówieniu tym nie zamieszczono zbyt wielu szczegółów na temat tego, w jaki sposób działają te technologie; nie omówiono także detali składni czy jej zastosowania. W rozdziale tym przechodzi się od razu do sedna sprawy i pokazuje przykład tego, w jaki sposób

aplikacje oparte na Ajaksie współpracują z każdą z trzech przedstawionych technologii, a także jak dana technologia po stronie serwera może przekazywać z powrotem do klienta informacje oraz dane, które mogą być użyte w aplikacji.

W niniejszym rozdziale omówiono następujące zagadnienia:

- Ajax oraz technologie po stronie serwera,
- ASP.NET,
- PHP,
- serwlety Javy.

Ajax oraz technologie po stronie serwera

W niniejszej książce zarysowano już najważniejszą różnicę pomiędzy sposobem działania tradycyjnych technologii po stronie serwera (model „dodaj dane-wyślij-czekaj") a sposobem działania aplikacji opartych na Ajaksie (asynchroniczne wywoływanie serwera), która polega na zmianie tradycyjnych wzorców użycia stron internetowych w aplikacjach ajaksowych.

Ajax nie jest ograniczony tylko do jednego wzorca użycia, a w rozdziale 9. zaprezentowane zostaną różne wzorce, w których Ajax może zmienić przepływ danych od klienta do serwera. Jedynym podobieństwem pomiędzy tymi wzorcami jest to, że nie jest się już przywiązanym do jawnego wysyłania formularza, a następnie zmuszonym do czekania na jego odpowiedź. Warto zatem skupić się chwilę na wykorzystywaniu formularzy oraz kontrolek HTML na stronie.

Formularze oraz kontrolki HTML

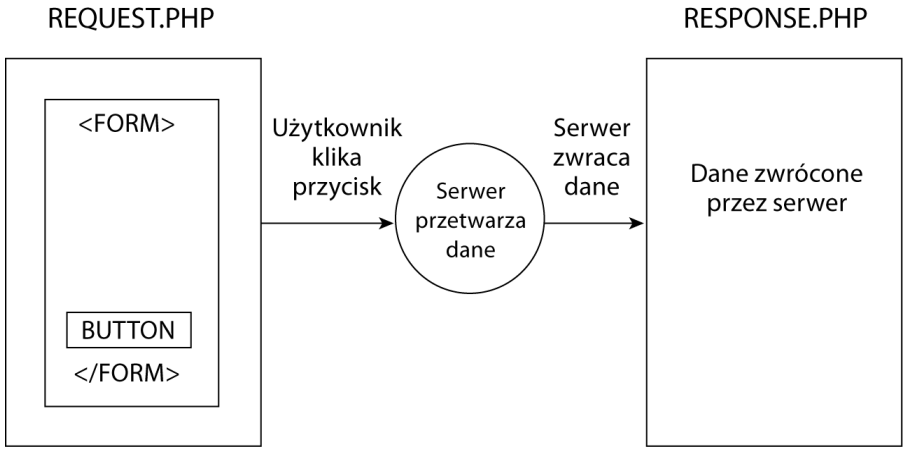
Ajax nie zmienia zasadniczego sposobu działania kontrolek HTML oraz sposobu pobierania z nich danych. Nadal ma się do dyspozycji zbiór kontrolek HTML (takich, jak listy rozwijane, przyciski opcji, pola wyboru czy pola tekstowe), które w dalszym ciągu są przesyłane do serwera. Serwer pobiera je w ten sam sposób.

Istnieją jednak dwie zasadnicze zmiany. Pierwszą z nich jest sposób wywoływania strony po stronie serwera, a drugą fakt, iż formularz HTML może być całkowicie usunięty ze strony bądź przerobiony tak, że nie działa w oczekiwany sposób. Zostanie to wyjaśnione po bliższym przyjrzeniu się temu, jak działa istniejący model przesyłania formularza do serwera.

Model przesyłania formularzy

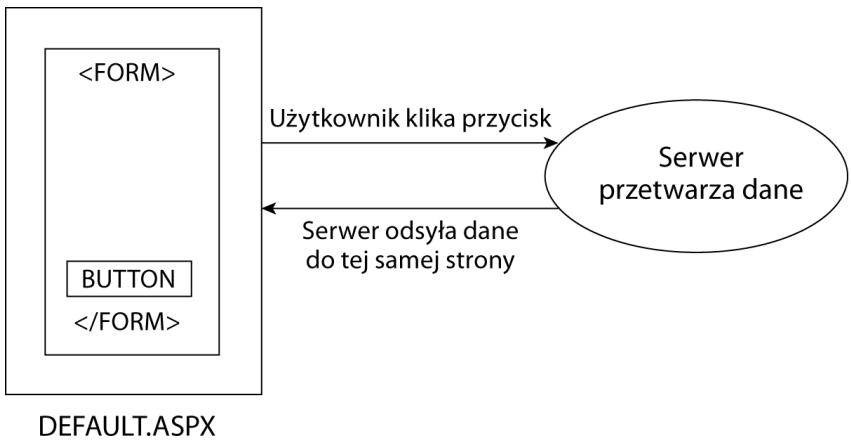
W przypadku dowolnej technologii po stronie serwera normalny sposób przesyłania formularza polega na dodaniu do niego przycisku, po kliknięciu którego formularz przesyłany jest do serwera.

Normalnie ASP czy PHP korzystają z atrybutu `action` formularza, by przekierować użytkownika ze strony początkowej na stronę odpowiedzi. Przetwarzanie wykonywane jest na serwerze, zanim użytkownik zostanie przekierowany do nowej strony, natomiast nowa strona używana jest do wyświetlenia danych (rysunek 3.1).



Rysunek 3.1. *Klasyczny model przesyłania formularzy w PHP oraz ASP*

W ASP.NET model ten nieco zmieniono, usuwając atrybut `action` i dodając atrybut `RUNAT="SERVER"`. Zamiast przechodzić do innej strony, strona ta prześle dane z powrotem do samej siebie. Ponownie jednak strona działa w momencie interwencji użytkownika (rysunek 3.2).



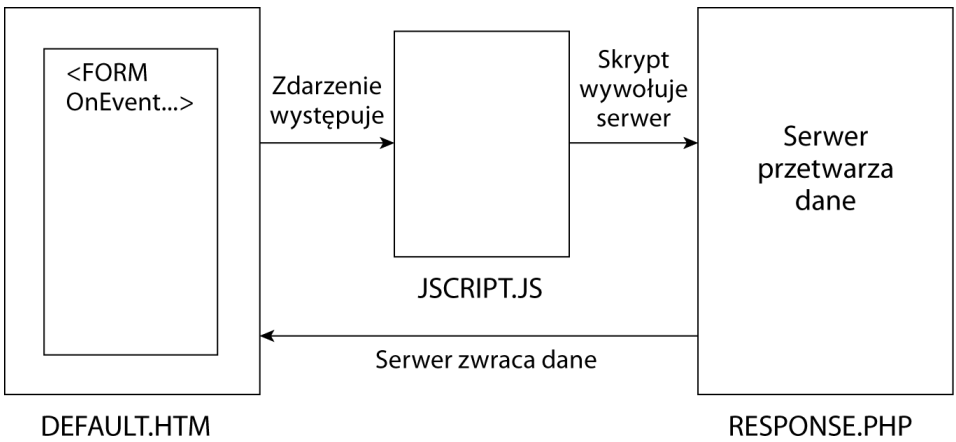
Rysunek 3.2. *Model przesyłania formularzy w ASP.NET*

W modelu ASP.NET nawigacja pomiędzy stronami wykonywana jest częściej za pomocą metod `Response.Redirect` bądź `Server.Transfer`. Jedną z największych przeszkód stojących przed początkującymi programistami przechodzącymi z ASP na ASP.NET było zrozumienie tej różnicy konceptualnej. Pozostało to popularnym zastosowaniem aż do czasów Ajaksa, który przywrócił pomysł wymuszenia na skrypcie przesyłania strony.

Model przesyłania formularzy w Ajaksie oraz JavaScripcie

W modelu przesyłania formularzy w JavaScripcie, który wykorzystywany jest w Ajaksie, znowu wprowadzono zmianę. Formularz może (choć nie musi) zostać całkowicie usunięty z tego modelu.

Model ten wykorzystuje JavaScript do przechwycenia wywołania zdarzenia, a kiedy takie zdarzenie następuje (na przykład gdy użytkownik kliknie przycisk bądź dokona wyboru w kontrolce HTML), wywołanie przekazywane jest do skryptu. To skrypt ma następnie zainicjować wywołanie serwera (rysunek 3.3).



Rysunek 3.3. *Model przesyłania formularzy w JavaScripcie oraz Ajaksie*

Skrypt nie musi oczywiście przesyłać danych natychmiast. Może poczekać na spełnienie innych warunków oraz kryteriów przed przesłaniem danych. W tym modelu — ponieważ skrypt może odpowiedzieć na zdarzenie natychmiast i nie musi czekać na jawne przesłanie — zwracanie danych z serwera nie musi również być natychmiast widoczne dla użytkownika. Skrypt nie jest ograniczony czekaniem na odpowiedź.

Od strony serwera

JavaScript wykorzystywany jest do rozpoczęcia interakcji pomiędzy klientem a serwerem. Zostawmy na razie ten podział i skupmy się na tym, co dzieje się z danymi, kiedy docierają one do serwera. Ponieważ każdy etap tego procesu odbywa się osobno, można podzielić go na części i każdą z nich omówić niezależnie od pozostałych. Części te można sobie wyobrazić jako czarne skrzynki. Wymagają one dostarczenia określonych danych wejściowych i dostarczają pewne dane wyjściowe, choć to, co dzieje się w środku, może być bardzo różne.

Rozważmy na przykład bankomat. Wkłada się do niego kartę, wpisuje numer identyfikacyjny (PIN) i oczekuje się, że pieniądze (oraz być może potwierdzenie) zostaną wydane przez maszynę. W całym kraju istnieją tysiące bankomatów należących do wielu różnych banków.

Niektóre pobierają opłaty, inne pozwalają sprawdzić szczegóły związane z kontem, a inne umożliwiają tylko dostęp do gotówki. Bankomaty działają różnie, jednak mają ten sam interfejs, za pomocą którego można się z nimi komunikować.

Część transakcji pomiędzy klientem a serwerem — odbywająca się po stronie serwera — przebiega dość podobnie jak w przypadku tej hipotetycznej czarnej skrzynki czy bankomatu. Otrzymuje ona dane z żądania HTTP i w końcu, po przetworzeniu owych danych, musi zwrócić je jako odpowiedź serwera.

Przesyłanie danych do serwera

Choć istnieje kilka metod przesyłania danych do serwera, skupimy się tylko na obiekcie `XMLHttpRequest`. By przesłać dane do serwera za pomocą tego obiektu, należy wykonać trzy kroki:

1. Ustawić zdarzenie, które zostanie wywołane po otrzymaniu danych.
2. Wywołać metodę `open` wraz z żądaniem.
3. Przesłać żądanie.

Metoda `open` jest jedyną, jaką trzeba będzie omówić.

```
XMLHttpRequestObject.open(metoda, URL do wywołania, asynchroniczna bądź
synchroniczna);
```

Metoda ta przesyła żądanie w jeden z dwóch możliwych sposobów. Pierwszy odbywa się za pośrednictwem metody HTTP `GET`, gdzie wiadomość przesyłana jest tak, jak w poniższym kodzie:

```
XMLHttpRequestObject.open("GET", "response.aspx?value=1", "true");
```

Drugi sposób to wykorzystanie metody HTTP `POST`. W poniższym fragmencie kodu dane ze zmiennej są kodowane w URI, opakowane w ciało żądania i przesyłane:

```
var argument = "value=";
argument += encodeURIComponent(dane);
XMLHttpRequestObject.open("POST", "response.aspx", "true");
xHRObject.send(ciało_żądania)
```

Kod ten pozwala na przesłanie wartości kontrolek HTML oraz danych z formularzy do serwera w dokładnie taki sam sposób i w tym samym formacie, w jakim przesyłano by stronę za pomocą PHP czy ASP.NET.

Serwer otrzymuje żądanie

Co dzieje się, kiedy serwer otrzymuje dane? Kod w JavaScripcie określił już metodę `GET` lub `POST`, która przesyła dane albo jako część adresu URL, albo jako część ciała żądania `Request`. W klasycznym ASP bądź w ASP.NET można wykorzystać zbiory `QueryString`, `Form` lub `Params` do pobrania tych elementów. W PHP używa się do tego zbiorów `$_GET`, `$_POST` bądź `$_REQUEST`.

Nie ma zauważalnej różnicy w otrzymywaniu danych za pomocą formularza HTML bądź z żądania pochodzącego ze skryptu w JavaScripcie. Można na przykład dołączyć do strony pole tekstowe o nazwie `MyTextbox1` dzięki następującemu kodowi na stronie HTML:

```
<input type="text" id="MyTextbox1" name="MyTextbox1" />
```

Kiedy wyśle się dane z pola za pomocą metody `GET` i z użyciem języka C#, dane pojawią się w zbiorze `QueryString` w ASP.NET:

```
string TextBox = Request.QueryString["MyTextbox1"].ToString();
```

Dane mogą też pojawić się w zbiorze `Form`, kiedy wysyłane są za pomocą metody `POST`.

```
string TextBox = Request.Form["MyTextbox1"].ToString();
```

Aby pobrać element o tej nazwie albo ze zbioru `QueryString`, albo z `Form`, można wykorzystać następujący kod:

```
string TextBox = Request.Params["MyTextbox1"].ToString();
```

W PHP odpowiednikiem tego będzie zbiór `$_REQUEST`. Można na przykład użyć następującego kodu:

```
$TextBox = $_REQUEST["MyTextbox1"];
```

W PHP można również pobrać wartości tych zbiorów osobno za pomocą zbiorów `$_GET` oraz `$_POST`, które są analogiczne do zbiorów `QueryString` oraz `Form` z ASP.NET.

```
$TextBox = $_GET["MyTextbox1"];

$TextBox = $_POST["MyTextbox1"];
```

Po pobraniu danych serwer może je przetworzyć i zwrócić do klienta.

Pisanie odpowiedzi HTTP

W normalnej sekwencji wydarzeń występującej przy użyciu technologii po stronie serwera informacje, które chce się wyświetlić, nie mogą być natychmiast wypisane na stronie. Zamiast tego trzeba spakować je w odpowiedź HTTP `Response`. Jest to o wiele łatwiejsze, niż może się wydawać na pierwszy rzut oka.

W ASP oraz ASP.NET wypisuje się dane za pomocą metody `Response.Write`.

```
string data = "To są nasze dane.";
Response.Write(data);
```

W PHP używa się polecenia `echo`.

```
$data = "To są nasze dane.";
echo $data
```

Można tworzyć bardziej skomplikowane struktury (takie jak dokumenty XML), pod warunkiem że będą analizowane składniowo jako tekst. Można to zaimplementować w ASP.NET w następujący sposób:

```
XmlDocument XmlDoc = new XmlDocument();
XmlNode versionNode = XmlDoc.CreateXmlDeclaration("1.0", "UTF-8", "yes");
XmlNode mainNode = XmlDoc.CreateElement("root");
XmlDoc.AppendChild(mainNode);
XmlNode childNode = XmlDoc.CreateElement("child");
childNode.AppendChild(XmlDoc.CreateTextNode("Dane"));
mainNode.AppendChild(versionNode);
mainNode.AppendChild(childNode);
string strXml = XmlDoc.InnerXml;
Response.Write(strXml);
```

W PHP to samo można zrobić w następujący sposób:

```
$doc = new DomDocument('1.0');
$root = $doc->createElement('root');
$root = $doc->appendChild('root');
$child = $doc->createElement('child');
$child = $root->appendChild($child);
$value = $doc->createTextNode("Dane");
$value = $child->appendChild($value);
$strXml = $doc->saveXML();
echo $strXml;
```

W ten sposób powstanie następujący dokument XML:

```
<?xml version="1.0"?>
<root>
<child>Dane</child>
</root>
```

Łańcuch znaków dołączany jest do odpowiedzi HTTP i odsyłany z powrotem do klienta jako gotowy do pobrania. Choć ta część procesu jest prosta, kiedy odpowiedź wraca do klienta, pobranie danych jest nieco bardziej skomplikowane.

Obiekt XMLHttpRequest

Jak zostanie to zaprezentowane w rozdziale 4., obiekt `XMLHttpRequest` nie jest jedyną metodą, którą ma się do dyspozycji w wykonywaniu interakcji pomiędzy klientem a serwerem w aplikacjach opartych na Ajaksie. Jest jednak metodą najczęściej używaną.

W niniejszym rozdziale nie omówiono szczegółów działania tego obiektu, dlatego najlepiej będzie go sobie wyobrazić jako kolejną czarną skrzynkę, która oczekuje na dane wejściowe z odpowiedzi HTTP.

Funkcja zwrotna

Pierwszy krok otrzymywania danych znany jest pod nazwą **funkcji zwrotnej** (ang. *callback function*). Jest to po prostu funkcja JavaScriptu, która wykonywana jest, kiedy dane zostaną w całości pobrane z serwera. Można jej nadać dość ogólną nazwę, taką jak na przykład `getData()`, a funkcja ta będzie w większości aplikacji opartych na Ajaksie wyglądała dość podobnie. Wewnątrz funkcji zwrotnej pierwszym zadaniem do wykonania jest sprawdzenie, czy dane są gotowe do pobrania. Wykonuje się to za pomocą sprawdzenia, czy właściwość `readyState` obiektu `XMLHttpRequest` równa jest `4` (wartość ta oznacza ukończenie). Na razie typowa funkcja zwrotna będzie wyglądała następująco:

```
function getData()
{
   if (xHRObject.readystate == 4)
   {
      // Tu następuje przetwarzanie
   }
}
```

Po upewnieniu się, że dane są gotowe, można je pobrać za pomocą jednej z dwóch właściwości obiektu `XMLHttpRequest`:

- `responseText`,
- `responseXML`.

Właściwość responseText

Wykorzystywanie właściwości `responseText` jest najczęściej stosowanym podejściem do pobierania danych z odpowiedzi HTTP; jest również najłatwiejszym rozwiązaniem. Można utworzyć nową zmienną JavaScriptu, która przechowa zawartość odpowiedzi i zwróci ją jako łańcuch znaków:

```
var text = xHRObject.responseText;
```

Jeśli na stronie w ASP.NET utworzy się poniższy kod, wtedy właściwość `responseText` dla `data` będzie zawierała tekst `To są nasze dane.`:

```
string data = "To są nasze dane.";
Response.Write(data);
```

W PHP należy zrobić to w następujący sposób:

```
$data = "To są nasze dane.";
echo $data
```

I to wszystko. Można również pobrać w ten sposób kod HTML bądź XHTML. Co się jednak stanie, gdy chce się pobrać dane XML? Nadal można użyć metody `responseText`, zwróci ona jednak dane XML w postaci łańcucha znaków. Można zastanowić się nad poniższym kodem w ASP.NET:

```
string data = "<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"yes\"?><root><child>Dane</child></root>";
Response.Write(data);
```

W PHP będzie to wyglądało następująco:

```
$data = "<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"yes\"?><root><child>Dane</child></root>";
echo $data
```

Powyższe fragmenty kodu zwrócą następujące dane:

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><root><child>Dane</child></root>
```

Zwracanie dokumentów XML w postaci łańcuchów znaków może jednak zniwelować wiele zalet korzystania z XML. Dlatego też istnieje właściwość `responseXML`.

Właściwość responseXML

Właściwość `responseXML` na pierwszy rzut oka wygląda, jakby lepiej nadawała się do sytuacji, w których chce się przekazywać z powrotem do klienta dokumenty XML. Pozwala ona traktować odpowiedź jako obiekt dokumentu XML i wykonywać iteracje po różnych elementach, atrybutach oraz węzłach tekstowych za pomocą DOM. Jak to jednak zwykle bywa w przypadku Ajaksa, trzeba sobie zdawać sprawę z istnienia kilku problemów związanych z korzystaniem z tej właściwości.

Powiedzmy, że mamy następujący kod po stronie serwera, który wczytuje dokument XML:

```
string data = "<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"yes\"?><root><child>Dane</child></root>";
Response.Write(data);
```

A kod w JavaScripcie zmieni się na następujący:

```
var text = xHRObject.responseXML;
```

Jeśli oczekuje się, że z powrotem otrzyma się w pełni działający dokument XML, to można się srogo zawieść. Z powrotem otrzyma się obiekt, jednak będzie on pusty, bez śladu po kodzie XML, jaki się przesyłało. Dzieje się tak, ponieważ `ContentType` odpowiedzi `Response` musi być ustawiony na `text/xml` przed wypisaniem odpowiedzi.

```
string data = "<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"yes\"?><root><child>Dane</child></root>";
Response.ContentType = "text/xml";
Response.Write(data);
```

Niestety, Internet Explorer szczególnie źle to toleruje. Jeśli nie ustawi się tego dobrze na serwerze, nie będzie można wykorzystywać `responseXML` w tej przeglądarce. W przeglądarce Firefox możliwe jest użycie w JavaScripcie metody `overrideMimeType` przed wywołaniem kodu, dzięki czemu można nadpisać i ustawić typ zawartości na `text/xml` po stronie klienta, jak w poniższym kodzie:

```
xHRObject.overrideMimeType("text/xml");
xHRObject.send(null);
var document = xHRObject.responseXML;
```

Metody tej nie ma jednak w przeglądarce Internet Explorer. Problemy nie kończą się jednak na tym. Jeśli popełni się błąd w dokumencie XML, tak że nie jest on poprawny składniowo, w IE otrzyma się ponownie pusty obiekt bez oczywistego komunikatu o błędzie.

Usuwanie błędów z responseXML

Przy ustalaniu przyczyny otrzymywania pustego obiektu z `responseXML` można użyć czterech metod. Pierwsza polega na sprawdzeniu, czy dane są zwracane w `responseText`. Można na przykład wykorzystać do tego okno dialogowe z ostrzeżeniem typu `alert`, jak poniżej:

```
var text = xHRObject.responseText;
alert(text);
```

Można oczekiwać, że zobaczy się coś takiego, jak poniżej:

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><root><child>Dane</child></root>
```

Jeśli tak nie jest, oznacza to, że odpowiedź nie została poprawnie przesłana przez serwer i należy sprawdzić kod po stronie serwera. Wysoce prawdopodobne jest jednak, że dane te są poprawne.

Jeśli dane nie są poprawne, kolejnym krokiem będzie sprawdzenie kodu błędu.

Usuwanie błędów z responseXML w przeglądarce Internet Explorer

By odnaleźć więcej informacji na temat błędu w przeglądarce Internet Explorer, należy skorzystać z poniższego kodu, by móc uzyskać nieco bardziej szczegółowy komunikat o błędzie oraz radę na temat tego, co należy poprawić w dokumencie XML:

```
var errorcode = xHRObject.responseXML.parseError.errorCode;
```

W normalnej sytuacji kod zwracany z przeglądarki Internet Explorer powinien być równy zero. Jednak, co bardziej prawdopodobne, jeśli `ContentType` na pewno jest ustawiony na `text/xml`, a `responseXML.xml` jest pusty, wtedy kod ten będzie inny niż zero.

Dalsze informacje na temat znaczenia kodu zwracanego z właściwości `responseXML` można uzyskać w następujący sposób:

```
var errormessage = xHRObject.responseXML.parseError.reason;
```

Usuwanie błędów z responseXML w przeglądarce Firefox

Choć w przeglądarce Firefox nie ma odpowiednika obiektu `parseError`, w menu *Narzędzia* można znaleźć opcję *Konsola błędów*, która pozwala na przejrzenie znaczących komunikatów o błędzie obejmujących typ obiektu powodujący wystąpienie błędu. Jeśli w odpowiedzi istnieje problem z formatowaniem XML, konsola udostępni informacje takie, jak poniższe, i nie wymaga to żadnych zmian w kodzie w JavaScripcie ani też dodawania do niego czegokolwiek:

```
Błąd: niepasujący znacznik. Oczekiwano: </error>.
Plik źródłowy:
http://localhost:8080/Company/WebPages/framed.jsf?com.asparity.AJAX_CLIENT_ID=
_idJsp0%3AmasterTree&com.company.AJAX_REQUEST=true&oracle.adf.faces.STATE_TOKEN=3&
nodeString=%3A%3AdomainModel%3A1&clientNodeId=%3A%3AdomainModel%3A1%3Aawaiting
AjaxData
Wiersz: 1, Kolumna: 6905
Kod źródłowy: [...]
```

Dodatkowo istnieje również łatwy do zainstalowania dodatek Firebug[1], który pozwala programistom badać ruch XHR w czasie rzeczywistym — zarówno żądania, jak i odpowiedzi.

Wykorzystywanie danych

Po zwróceniu danych z właściwości `responseXML` można je pobrać tak, jakby były obiektem DOM. Załóżmy na przykład, że mamy następujący dokument:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cart>
   <book>
      <Title>Ajax. Od podstaw.</Title>
      <Quantity>1</Quantity>
   </book>
</cart>
```

Można zwrócić element `<cart>` z dokumentu XML w następujący sposób:

```
var XMLDoc = xHRObject.responseXML;
var book = XMLDoc.getElementsByTagName("book");
```

Można również przejść do pierwszego elementu zawartego w `<book>` w następujący sposób:

```
var title = book[0].firstChild;
```

Istnieje znacząca różnica pomiędzy dwoma najważniejszymi przeglądarkami (Internet Explorer oraz Mozilla) w kwestii tego, w jaki sposób zawartość tekstowa zwracana jest z dokumentów XML.

W Internet Explorerze zawartość tekstowa zwracana jest za pomocą właściwości `text`, jak w poniższym kodzie:

```
var title = book[0].firstChild.text;
// Tytuł będzie równy "Ajax. Od podstaw."
```

W przeglądarkach Mozilla zawartość tekstową zwraca się za pomocą właściwości `textContent`, co widać w poniższym fragmencie kodu:

```
var title = book[1].firstChild.textContent;
// Tytuł będzie równy "Ajax. Od podstaw."
```

[1] Firebug, a także inne narzędzia i sposoby usuwania błędów z aplikacji, omówione są w rozdziale 6. — *przyp. tłum.*

Nietrudno również zauważyć, że w Internet Explorerze do pierwszego węzła odnosi się przez `book[0]`, natomiast w przypadku przeglądarki Firefox jest to `book[1]`. Dzieje się tak, ponieważ w Firefoksie `book[0]` zawiera węzeł tekstowy ze znakiem nowego wiersza, gdyż przeglądarka ta nie opuszcza białych znaków (ang. *whitespace*) — traktuje je jak osobne węzły, podczas gdy Internet Explorer tego nie robi.

I znów niezbędne jest uwzględnienie tych różnic w kodzie, by dostęp do danych zwracanych z serwera działał we wszystkich przeglądarkach.

Technologie po stronie serwera

Dotychczas wspomniano jedynie o procesie, w którym dane mogą być przesyłane do serwera i odsyłane z niego; nie omawiano tego, co dzieje się na samym serwerze. Technologie po stronie serwera to zagadnienie niezależne od Ajaksa, dlatego trzeba się tych kwestii uczyć osobno. Załóżmy zatem, że Czytelnik posiada już praktyczną znajomość jednej z tych technologii, ponieważ bez umiejętności wykonywania przetwarzania na serwerze daleko się nie zajdzie. Poniżej znajduje się krótkie wprowadzenie do każdej z wybranych technologii wraz z prostą przykładową aplikacją opartą na Ajaksie, która z nich korzysta.

Jeśli Czytelnik jest zaznajomiony z ASP.NET, ale z PHP czy Javą już nie, nie powinien pomijać podrozdziałów poświęconych tym ostatnim językom. Znajomość innych języków czy technologii poza tymi, które już się dobrze zna, daje programiście nie tylko dobre podstawy, ale także zabezpieczenie na przyszłość. Często można spotkać kontrakty czy zlecenia, które obejmują przeniesienie aplikacji z jednego języka na drugi, co wymaga dobrej znajomości obu.

Klasyczny ASP oraz PHP są do siebie dość podobne i wkrótce ich podobieństwo zostanie zaprezentowane. W poniższym omówieniu zostanie krótko wspomniana Java, by pokazać, w jaki sposób zgrabnie łączy się ona z filozofią Ajaksa. Na początek jednak pora na aktualny sztandarowy produkt firmy Microsoft — ASP.NET.

ASP.NET

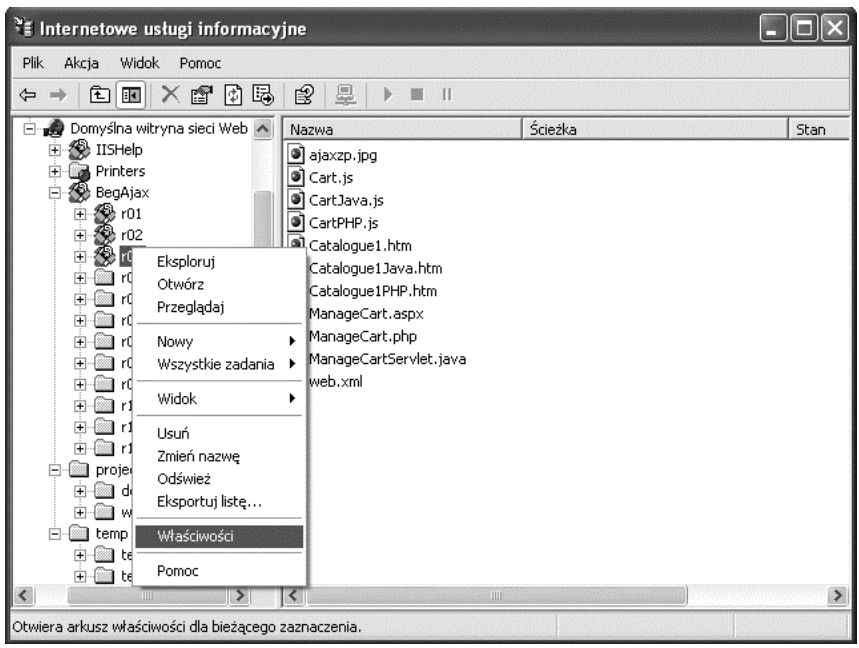
ASP.NET to technologia firmy Microsoft służąca tworzeniu dynamicznych stron internetowych. By mogła działać na komputerze, konieczne jest spełnienie dwóch warunków: zainstalowanie platformy .NET Framework oraz posiadanie kompatybilnego serwera WWW, najczęściej IIS (Internet Information Services).

.NET istnieje od 2002 roku, jednak aktualne większe wydanie (.NET 3.0) zostało opublikowane w listopadzie 2006 roku. Najnowszą wersję można pobrać ze strony *http://update.microsoft.com*. Należy wybrać opcję *Instalacja niestandardowa*, a nie *Instalacja ekspresowa*, i szukać w *Oprogramowanie opcjonalne*.

Serwer IIS jest z kolei dostępny tylko jako część systemu operacyjnego Windows. W wielu wersjach systemu Windows domyślnie nie jest on zainstalowany i tym samym nie jest dostępny w Windows XP Home Edition. Można go jednak dodać, przechodząc do *Panelu sterowania* i wybierając *Dodaj lub usuń programy*, a później *Dodaj/Usuń składniki systemu Windows*.

ASP.NET ma być niezależny od języka programowania. Pomysł polega na tym, że można używać implementacji dowolnego języka w .NET, w którym tworzy się aplikacje. Zazwyczaj sprowadza się to do wyboru pomiędzy dwoma językami programowania — Visual Basic oraz C#. W tej książce przykłady w ASP.NET tworzone będą w języku C#, ponieważ jest on podobny do JavaScriptu, jeśli chodzi o strukturę oraz opcje, takie jak znaczenie wielkości liter. Język ten jest również bardziej zbliżony do PHP.

By uruchomić aplikację w ASP.NET, należy umieścić ją w folderze *C:\inetpub\wwwroot\nazwa_aplikacji* (czy też dowolnym innym miejscu wybranym dla aplikacji), a następnie użyć serwera IIS do utworzenia wirtualnego katalogu. Można to zrobić poprzez uruchomienie IIS i kliknięcie prawym przyciskiem myszy katalogu, jaki chce się uruchomić, a następnie wybranie jego *Właściwości*, co widać na rysunku 3.4.



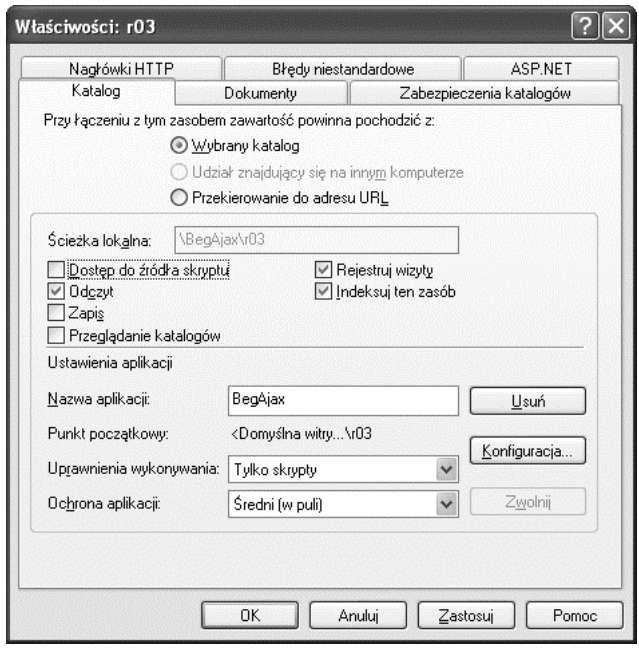
Rysunek 3.4. *Wybieranie właściwości katalogu w IIS*

Następnie w oknie dialogowym, które się pokaże, należy kliknąć przycisk *Utwórz*, dzięki czemu tworzy się nową aplikację (jak widać na rysunku 3.5).

Później w przeglądarce przechodzi się do nowej aplikacji (w tym przypadku do *http://localhost/BegAjax/r03*), po czym otrzymuje się *Błąd HTTP 403 — Dostęp zabroniony*, który oznacza, że udało się utworzyć katalog.

Po poprawnym umieszczeniu aplikacji na serwerze WWW wywołuje się ją z JavaScriptu dzięki odniesieniu się do strony ASP.NET, jaką chce się wywołać, za pomocą metody `Open` obiektu `XMLHttpRequest`, jak poniżej:

```
XMLHttpRequestObject.open("POST", "response.aspx?value=1", "true");
```



Rysunek 3.5. *Przycisk Utwórz w oknie dialogowym Właściwości*

Przykład wykorzystujący Ajaksa oraz ASP.NET

Utwórzmy fikcyjną stronę z katalogiem dla sprzedawcy książek wraz z koszykiem na zakupy. Pozwoli to użytkownikowi na umieszczanie wybranych przedmiotów w koszyku z zakupami i uaktualnianie tego koszyka bez konieczności odświeżania strony. Załóżmy, że użytkownik został już zidentyfikowany, by strona mogła być tak prosta, jak to tylko możliwe.

Koszyk z zakupami będzie miał trzy opcje:

- można do niego dodawać przedmioty,
- jeśli doda się do niego kolejny przedmiot, liczba przedmiotów wzrośnie o jeden,
- można z niego usuwać przedmioty.

Oczywiście nie jest to coś, co można osiągnąć całkowicie po stronie klienta, choć po stronie klienta wszystko można zainicjować. Aplikacja odczyta tytuł książki ze strony klienta i przekaże go do serwera. Jest to wykorzystywane przy identyfikacji książki, kiedy umieszcza się ją w koszyku z zakupami.

Na serwerze koszyk z zakupami musi śledzić, jakie przedmioty umieścił w nim użytkownik, a także jaka jest ich liczba. By móc tego dokonać, koszyk zostanie przechowany w zmiennej sesji. Oznacza to, że w dużym katalogu użytkownik będzie mógł przechodzić ze strony do strony bez tracenia informacji o tym, co zostało umieszczone w koszyku.

Do przechowania informacji w koszyku z zakupami można wykorzystać dokument XML. Istnieje tylko jeden element koszyka, a wewnątrz niego znajduje się element książki dla każdego tytułu, jaki kupuje użytkownik. Element książki zawiera elementy tytułu oraz liczby

egzemplarzy (choć można oczywiście dodać numer ISBN, cenę oraz autorów). Choć ma się tylko jedną stronę katalogu, przykład ten jest skalowalny, więc powinno się go dać uruchomić na wielu stronach z katalogami produktów.

spróbuj sam Przykład z koszykiem z zakupami w ASP.NET

1. Należy utworzyć nowy plik o nazwie *Catalogue.htm*.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
   <head>
      <script type="text/javascript" src="Cart.js"></script>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
   </head>
   <body>
      <br/>
      <img id="cover" src="ajaxzp.jpg" />
      <br />
      <br />
      <b>Książka:</b><span id="book">Ajax. Zaawansowane programowanie</span><br />
      <b>Autor: </b><span id="authors">Nicholas C. Zakas, Jeremy McPeak,
      Joe Fawcett</span>
      <br /><b>ISBN: </b><span id="ISBN">978-83-246-0567-5</span>
      <br /><b>Cena: </b><span id="price">67.00</span>
      <br /><br />
      <a href="#" onclick="AddRemoveItem('Add');" >Dodaj do koszyka</a>
      <br /><br />
      <span id="cart" ></span>
   </body>
</html>
```

1. Należy utworzyć skrypt o nazwie *Cart.js*.

```
var xHRObject = false;
if (window.XMLHttpRequest)
{
   xHRObject = new XMLHttpRequest();
}
   else if (window.ActiveXObject)
{
   xHRObject = new ActiveXObject("Microsoft.XMLHTTP");
}

function getData()
{
   if ((xHRObject.readyState == 4) && (xHRObject.status == 200))
   {
      var serverResponse = xHRObject.responseXML;
      var header = serverResponse.getElementsByTagName("book");
      var spantag = document.getElementById("cart");
      spantag.innerHTML = "";
      for (i=0; i<header.length; i++)
      {
         if (window.ActiveXObject)
         {
            spantag.innerHTML += " " +header[0].firstChild.text;
```

```
            spantag.innerHTML += " " + header[0].lastChild.text + " " + "<a
            href='#' onclick='AddRemoveItem(\"Remove\");'>Usuń przedmiot</a>";
         }
         else
         {
            spantag.innerHTML += " " +header[0].firstChild.textContent;
            spantag.innerHTML += " " + header[0].lastChild.textContent + " " + "<a
            href='#' onclick='AddRemoveItem(\"Remove\");'>Usuń przedmiot</a>";
         }
      }
   }
}

function AddRemoveItem(action)
{
   var book  = document.getElementById("book").innerHTML;
   if(action=="Add")
   {
      xHRObject.open("GET", "ManageCart.aspx?action=" + action + "&book=" +
      encodeURIComponent(book) + "&value=" + Number(new Date), true);
   }
   else
   {
      xHRObject.open("GET", "ManageCart.aspx?action=" + action + "&book=" +
      encodeURIComponent(book) + "&value="  + Number(new Date), true);
   }
   xHRObject.onreadystatechange = getData;
   xHRObject.send(null);
}
```

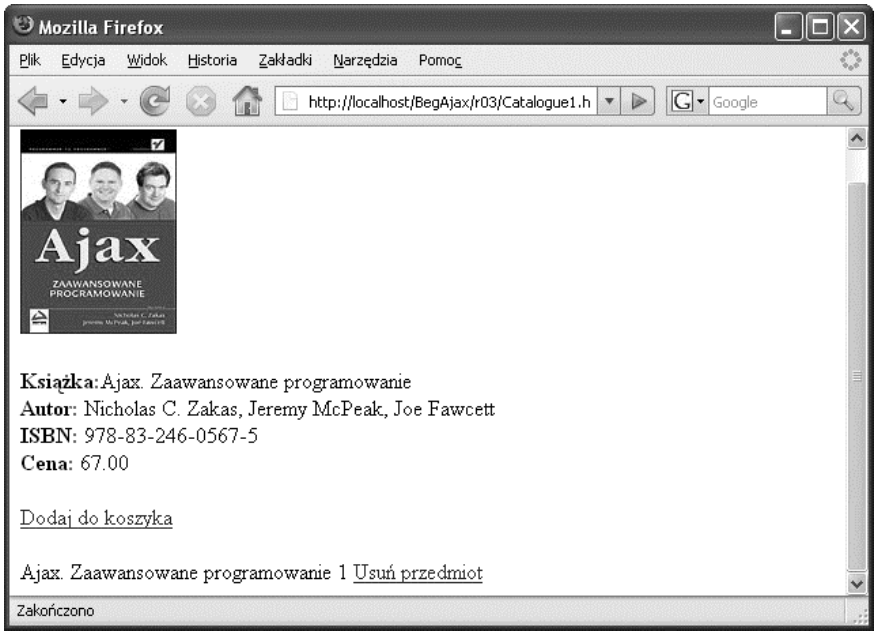
1. Należy utworzyć stronę o nazwie *ManageCart.aspx*.

```
<%@Page Language = "C#" Debug="true" %>

<%@ import Namespace="System.Xml" %>
<script language="C#" runat="server">
   void Page_Load()
   {
      string newitem = Request.Params["book"];
      string action = Request.Params["action"];
      Hashtable ht = new Hashtable();
      if (Session["Cart"] != null)
      {
         ht = (Hashtable)Session["Cart"];
         if (action == "Add")
         {
            if (ht.ContainsKey(newitem))
            {
               int value = int.Parse(ht[newitem].ToString());
               ht.Remove(newitem);
               value++;
               ht.Add(newitem, value);
               Session["Cart"] = ht;
               Response.ContentType = "text/xml";
               Response.Write(toXml(ht));
            }
```

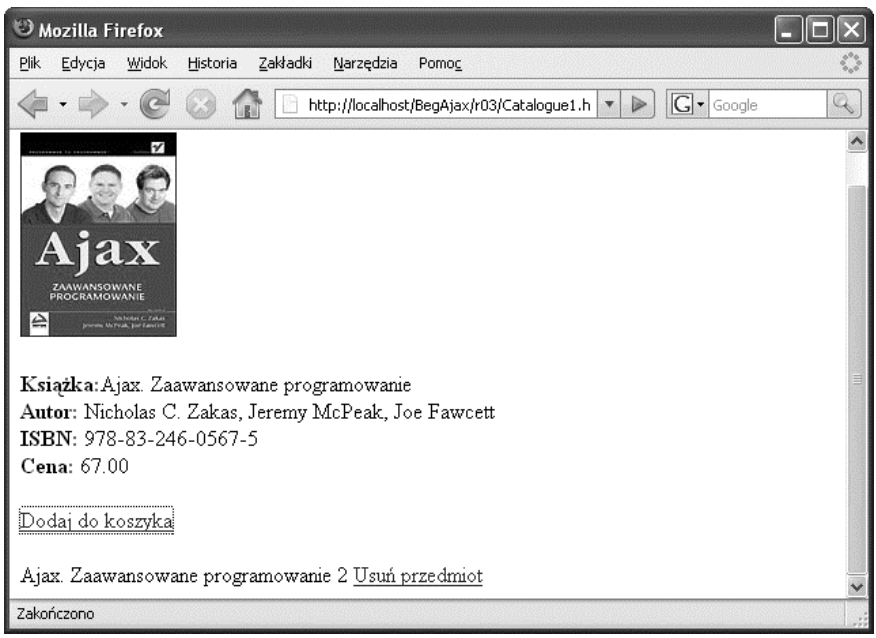
```
            else
            {
               ht.Add(newitem, 1);
               Session["Cart"] = ht;
               Response.ContentType = "text/xml";
               Response.Write(toXml(ht));
            }
         }
         else
         {
            ht.Remove(newitem);
            Session["Cart"] = null;
            Response.ContentType = "text/xml";
            Response.Write(toXml(ht));
         }
      }
      else
      {
         ht.Add(newitem, 1);
         Session["Cart"] = ht;
         Response.ContentType = "text/xml";
         Response.Write(toXml(ht));
      }
   }

   string toXml(Hashtable ht)
   {
      XmlDocument XmlDoc = new XmlDocument();
      XmlNode versionNode = XmlDoc.CreateXmlDeclaration("1.0","UTF-8","yes");
      XmlNode mainNode = XmlDoc.CreateElement("cart");
      XmlDoc.AppendChild(versionNode);
      XmlDoc.AppendChild(mainNode);
      foreach (string key in ht.Keys)
      {
         XmlNode childNode = XmlDoc.CreateElement("book");
         XmlNode TitleNode = XmlDoc.CreateElement("Title");
         XmlNode QuantityNode = XmlDoc.CreateElement("Quantity");
         TitleNode.AppendChild(XmlDoc.CreateTextNode(key));
         QuantityNode.AppendChild(XmlDoc.CreateTextNode(ht[key].ToString()));
         childNode.AppendChild(TitleNode);
         childNode.AppendChild(QuantityNode);
         mainNode.AppendChild(childNode);
      }
      string strXml = XmlDoc.InnerXml;
      return strXml;
   }
</script>
```

1. Należy uruchomić plik *Catalogue1.htm* w przeglądarce, a następnie wybrać odnośnik *Dodaj do koszyka*, jak widać na rysunku 3.6.

5. Należy kliknąć ponownie odnośnik *Dodaj do koszyka*, a liczba przedmiotów w koszyku z zakupami wzrośnie o jeden, co widać na rysunku 3.7.

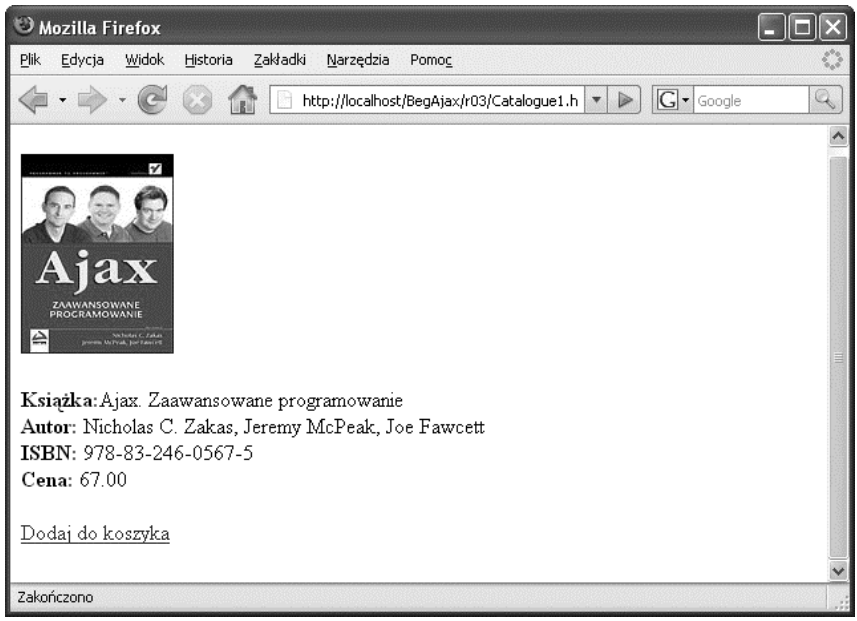


Rysunek 3.6. *Odnośnik Dodaj do koszyka*



Rysunek 3.7. *Liczba przedmiotów w koszyku zwiększa się o jeden*

6. Teraz należy kliknąć odnośnik *Usuń przedmiot* — przedmiot znika z koszyka z zakupami, co widać na rysunku 3.8.



Rysunek 3.8. *Przedmiot znika z koszyka z zakupami*

Jak to działa

Cykl trzech stron dokładnie odzwierciedla model JavaScriptu omówiony wcześniej. Strona HTML jest „sztuczną" stroną, na której wyświetlany jest jeden przedmiot z katalogu. W rzeczywistości byłaby to strona po stronie serwera, która pobierałaby szczegóły na temat książki z bazy danych i wyświetlała je na ekranie. Strona z katalogu zawiera również skrypt *Cart.js*. Skrypt ten wywoływany jest za pomocą odnośnika *Dodaj do koszyka*:

```
<a href="#" onclick="AddRemoveItem('Add');" >Dodaj do koszyka</a>
```

W skrypcie wykorzystuje się jedną funkcję zarówno do dodawania do koszyka, jak i do usuwania z niego przedmiotów. Do skryptu jako argument przekazuje się tylko zmienną będącą łańcuchem znaków, wskazując na to, czy chce się przedmiot dodać (`Add`), czy usunąć (`Remove`). Nie ma co skupiać się na szczegółach tej funkcji, ponieważ omówienie obiektu `XMLHttpRequest` nie jest celem tego ćwiczenia.

Funkcja dokonuje ekstrakcji książki ze strony za pomocą metody `document.getElementById` i tworzy jedno z dwóch zapytań do serwera — w zależności od tego, czy element chce się dodać do koszyka (`Add`), czy z niego usunąć (`Remove`). Składa ona łańcuch znaków zapytania i wywołuje stronę po stronie serwera.

```
function AddRemoveItem(action)
{
   var book  = document.getElementById("book").innerHTML;
   if(action=="Add")
   {
      xHRObject.open("GET", "ManageCart.aspx?action=" + action + "&book=" +
      encodeURIComponent(book) + "&value=" + Number(new Date), true);
   }
   else
   {
      xHRObject.open("GET", "ManageCart.aspx?action=" + action + "&book=" +
      encodeURIComponent(book) + "&value="  + Number(new Date), true);
   }
   xHRObject.onreadystatechange = getData;
   xHRObject.send(null);
}
```

W kodzie po stronie serwera *ManageCart.aspx* istnieją dwie metody. Pierwsza część znajduje się w metodzie `Page_Load` i wykonywana jest po wywołaniu strony. Druga — `toXml` — serializuje koszyk z zakupami do postaci XML i przygotowuje go do odesłania z powrotem do klienta. Rozpoczyna się w `Page_Load` od utworzenia trzech zmiennych. Pierwsze dwie służą do przechowywania tytułu książki pobranego ze strony znajdującej się po stronie klienta, natomiast trzecia jest działaniem, jakie wybrał użytkownik (może to być `Add` bądź `Remove`). Następnie tworzy się tablicę mieszającą (ang. *hash table*), w której przechowywane będą przedmioty z koszyka z zakupami. W koszyku należy przechować tytuł przedmiotu wraz z liczbą egzemplarzy, jaką wybrał użytkownik.

```
void Page_Load()
{
   string newitem = Request.Params["book"];
   string action = Request.Params["action"];
   Hashtable ht = new Hashtable();
```

Następnie sprawdza się, czy zmienna `Session` przechowująca koszyk z zakupami jest pusta. Jeśli nie jest pusta, wiadomo, że użytkownik już odwiedzał stronę, dlatego może to być jedna z poniższych sytuacji:

- Użytkownik dodaje przedmiot i przedmiot ten jest już obecny w koszyku z zakupami.
- Użytkownik dodaje nowy przedmiot, którego nie ma jeszcze w koszyku z zakupami, natomiast sam koszyk z zakupami już istnieje.
- Użytkownik usuwa przedmiot z koszyka z zakupami.
- Użytkownik dodaje przedmiot, ale koszyk jeszcze nie istnieje, dlatego trzeba go utworzyć.

Jeśli użytkownik dodaje przedmiot do koszyka z zakupami, a przedmiot ten jest już obecny w `Hashtable ht`, można wykonać następujący kod:

```
   if (Session["Cart"] != null)
   {
      ht = (Hashtable)Session["Cart"];
      if (action == "Add")
```

```
        {
            if (ht.ContainsKey(newitem))
            {
                int value = int.Parse(ht[newitem].ToString());
                ht.Remove(newitem);
                value++;
                ht.Add(newitem, value);
                Session["Cart"] = ht;
                Response.ContentType = "text/xml";
                Response.Write(toXml(ht));
            }
```

Bez względu na wykonywane działanie należy się upewnić, że otrzyma się najbardziej aktualną wersję koszyka z zakupami ze zmiennej `Session`, a następnie przechowa się ją w `Hashtable ht`. Sprawdza się, czy zmienna działania równa jest `Add`, a także to, czy przedmiot obecny jest w tablicy mieszającej. Następnie tworzy się wartość zmiennej, która przechowuje liczbę powiązaną z przedmiotem znajdującym się w tablicy mieszającej. Przedmiot usuwany jest z tej tablicy. Dodaje się jeden do wartości, a potem dodaje się nowy przedmiot do tablicy mieszającej. Wczytuje się uaktualnioną tablicę do zmiennej `Session`.

Kolejny krok jest bardzo istotny. Ustawia się `ContentType` na `text/xml`. Jeśli się tego nie zrobi, właściwość `responseXML` nie będzie działała poprawnie. Na koniec zapisuje się tablicę mieszającą do strumienia `Response` za pomocą funkcji `toXml` (co zostanie omówione wkrótce).

Jeśli tablica mieszająca nie zawiera przedmiotu, należy go dodać, zatem wykonuje się poniższy kod:

```
            else
            {
                ht.Add(newitem, 1);
                Session["Cart"] = ht;
                Response.ContentType = "text/xml";
                Response.Write(toXml(ht));
            }
        }
```

Przedmiot dodaje się do tablicy mieszającej i przypisuje się mu wartość `1`. Ustawia się również `ContentType` na `text/xml`, a następnie zapisuje wszystko do strumienia `Response`. Trzeba to zrobić na końcu każdego możliwego scenariusza, by upewnić się, że klient wyświetla poprawne informacje.

W trzecim scenariuszu zmienna `Action` nie była równa `Add`, dlatego zakłada się, że musi mieć wartość `Remove`. By usunąć przedmiot, należy wykonać następujący kod:

```
        else
        {
            ht.Remove(newitem);
            Session["Cart"] = null;
            Response.ContentType = "text/xml";
            Response.Write(toXml(ht));
        }
    }
```

Przedmiot usuwany jest z tablicy mieszającej. Ustawia się zmienną `Session` zawierającą koszyk z zakupami na zero, po czym wartość `ContentType` ustawiana jest na `text/xml`, a tablica mieszająca zapisywana jest do strumienia `Response`.

Ostatni scenariusz będzie wyglądał znajomo. Jest on w zasadzie prawie taki sam jak drugi. Dodaje się przedmiot do tablicy mieszającej, przechowuje się tablicę mieszającą w zmiennej `Session`, zapisuje `ContentType` jako `text/xml` i zapisuje tablicę mieszającą do strumienia `Response`.

```
else
{
   ht.Add(newitem, 1);
   Session["Cart"] = ht;
   Response.ContentType = "text/xml";
   Response.Write(toXml(ht));

}
```

Na końcu każdego scenariusza wywołuje się funkcję `toXml`, która serializuje tablicę mieszającą do postaci XML. Więcej informacji na temat XML znajduje się w rozdziale 5., dlatego nie będziemy teraz tracić czasu na omawianie struktury dokumentu.

```
XmlDocument XmlDoc = new XmlDocument();
XmlNode versionNode = XmlDoc.CreateXmlDeclaration("1.0","UTF-8","yes");
XmlNode mainNode = XmlDoc.CreateElement("cart");
XmlDoc.AppendChild(versionNode);
XmlDoc.AppendChild(mainNode);
```

Tworzy się nowy dokument, a także jego węzeł deklaracji. Tworzy się także węzeł główny dokumentu i nadaje mu nazwę `cart`. Następnie dodaje się deklarację oraz węzeł główny do dokumentu.

Później wykonuje się iterację po każdym kluczu z tablicy mieszającej. Przechowano tytuł książki (`Title`) jako klucz w tablicy, a także liczbę egzemplarzy, jaką użytkownik chce zakupić (`Quantity`), jako wartość. Dla każdego elementu z tablicy mieszającej tworzy się element książki z `Title` oraz `Quantity`. Ponieważ `Title` oraz `Quantity` są podelementami książki, należy uważać, w jakiej kolejności się je dodaje i do jakich elementów.

```
foreach (string key in ht.Keys)
{
   XmlNode childNode = XmlDoc.CreateElement("book");
   XmlNode TitleNode = XmlDoc.CreateElement("Title");
   XmlNode QuantityNode = XmlDoc.CreateElement("Quantity");
```

Najpierw jednak należy dodać wartości tekstowe z tablicy mieszającej do elementów. Dodaje się klucz tablicy do elementu `Title`, a wartość tego klucza do elementu `Quantity`:

```
   TitleNode.AppendChild(XmlDoc.CreateTextNode(key));
   QuantityNode.AppendChild(XmlDoc.CreateTextNode(ht[key].ToString()));
```

Następnie dodaje się dwa podelementy do elementu potomnego `<book>`. Na koniec dodaje się książkę do elementu `<cart>`, co kończy dokument.

```
   childNode.AppendChild(TitleNode);
   childNode.AppendChild(QuantityNode);
   mainNode.AppendChild(childNode);
}
```

Jedyne, co trzeba jeszcze zrobić, to otrzymanie tekstowej wersji tego dokumentu za pomocą metody `innerXml` i zwrócenie jej do metody wywołującej:

```
string strXml = XmlDoc.InnerXml;

return strXml;
```

Oczywiście nie wszystko jest jeszcze gotowe. Udało się przesłać dane do serwera, natomiast na serwerze spakowano je i wysłano odpowiedź z powrotem w postaci dokumentu XML. Trzeba zatem odczytać dokument XML i wyświetlić go na stronie. Jest to zadanie dla JavaScriptu oraz funkcji `getData()`, która jest wywoływana po wywołaniu zdarzenia `readystatechange`.

Wewnątrz funkcji zwrotnej pobiera się dane z właściwości `responseXML` obiektu `XMLHttpRequest`. Izoluje się podzbiór XML za pomocą metody DOM `getElementsByTagName`, by pozyskać tablicę elementów będących książkami.

```
var serverResponse = xHRObject.responseXML;
var header = serverResponse.getElementsByTagName("book");
```

W tym samym czasie trzeba również pobrać informacje o elemencie HTML, który będzie wyświetlał koszyk z zakupami. Na stronie *Catalogue1.htm* utworzono pusty element `<span>`. Tutaj pobiera się go za pomocą metody DOM `getElementById` i upewnia się, że jego właściwość `innerHTML` ustawiona jest na zero.

```
var spantag = document.getElementById("cart");
spantag.innerHTML = "";
```

Kiedy pobrano już wszystkie elementy książki i przechowano je w tablicy, czas je rozpakować. Tworzy się pętlę i ustawia jej maksimum na liczbę elementów w tablicy. Następnie sprawdza się, czy użytkownik korzysta z przeglądarki Internet Explorer, czy Mozilla. Jeśli używa Internet Explorera, należy skorzystać z właściwości `text` do wyświetlenia każdego elementu książki. Jeśli przeglądarka wykorzystywana przez użytkownika to Mozilla, wtedy należy użyć właściwość `textContent`. Pierwszym elementem potomnym (`firstChild`) elementu książki jest `Title`. Ostatnim elementem potomnym (`lastChild`) jest `Quantity`.

```
for (i=0; i<header.length; i++)
{
   if (window.ActiveXObject)
   {
      spantag.innerHTML += " " +header[0].firstChild.text;
      spantag.innerHTML += " " + header[0].lastChild.text + " " + "<a href='#'
      onclick='AddRemoveItem(\"Remove\");'>Usuń przedmiot</a>";
   }
   else
   {
      spantag.innerHTML += " " +header[0].firstChild.textContent;
      spantag.innerHTML += " " + header[0].lastChild.textContent + " " +
      "<a href='#' onclick='AddRemoveItem(\"Remove\");'>Usuń przedmiot</a>";
   }
}
```

Po wyświetleniu tytułu oraz liczby wybranych książek tworzy się również dynamiczny odnośnik pozwalający użytkownikowi usunąć przedmiot z koszyka. Odnosi się on z powrotem do JavaScriptu z wywołaniem `AddRemoveItem`, przekazując do tej funkcji `Remove` jako argument. W ten sposób cykl zostaje zakończony.

Strona ASP.NET zawiera dość sporo kodu, jednak, jak widać, w samym kodzie nie ma nic szczególnie skomplikowanego. Tak samo będzie w przypadku wersji w PHP.

PHP

PHP jest akronimem, o którym mówi się, że ma kilka znaczeń. Najczęściej akceptowanym jest po prostu *Hypertext PreProcessor*, choć dla innych będzie to *Personal Home Pages* czy *PHP Hypertext PreProcessor* (potencjalny akronim rekursywny). PHP jest chyba najprostszą alternatywą dla ASP.NET. W przeciwieństwie do ASP.NET (który wykorzystuje istniejące języki programowania) PHP sam w sobie jest językiem programowania. Może działać na najważniejszych serwerach WWW (IIS oraz Apache) i nie jest powiązany z platformą Windows. W języku tym udało się zachować wiele z prostoty, która na początku przyciągnęła tylu programistów do klasycznego ASP, to znaczy język ten nie wymaga silnego typowania zmiennych ani też pakowania wszystkiego w osobne procedury bądź metody.

PHP jest produktem Open Source, który został skomercjalizowany przez firmę Zend Technologies. Można go pobrać ze strony *www.php.net*. Aktualną wersją w momencie pisania niniejszej książki jest 5.1.6. Podczas gdy ASP.NET teoretycznie wymaga przynajmniej zakupu systemu operacyjnego Windows, PHP jest zupełnie darmowy, ponieważ może działać na systemie operacyjnym z gatunku Open Source (takim jak Linux). Działa również na popularnym serwerze WWW Apache (*www.apache.org*), który może się pochwalić tak samo szeroką obsługą różnych platform oraz elastycznością jak sam PHP.

PHP przypomina dwa główne języki. Został utworzony w języku PERL przez Rasmusa Lerdorfa, a także w języku C. Jest jednak w dużej mierze pozbawiony stopnia skomplikowania związanego z tymi językami. Ma zbiór obiektów, który naśladuje obiekty klasycznego ASP oraz ASP.NET, włącznie z obiektami `Request` oraz `Session`, dlatego nie powinien wyglądać zbyt obco dla programistów pracujących w tych dwóch językach.

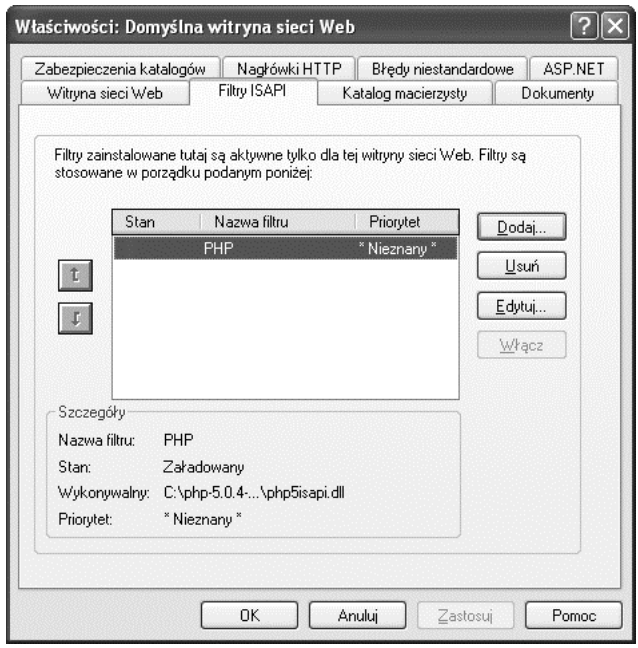
Choć poprzednie wcielenia języka PHP cieszyły się opinią niełatwych w instalacji, najnowsza wersja PHP 5 jest bardzo łatwa w użyciu na serwerze IIS. Jeśli utworzy się katalog na twardym dysku i dokona ekstrakcji pliku instalacyjnego ZIP dla Windows pobranego ze strony *www.php.net/downloads.php#v5*, wystarczy tylko zarejestrować filtr ISAPI za pomocą IIS. Żeby tego dokonać, należy przejść do IIS, kliknąć prawym przyciskiem myszy pozycję w *Domyślna witryna sieci Web* i wybrać *Właściwości*. Należy wybrać zakładkę *Filtry ISAPI*, pokazaną na rysunku 3.9. Trzeba kliknąć *Dodaj*, utworzyć filtr o nazwie PHP i skierować plik wykonywalny na *php5isapi.dll*.

Po poprawnej instalacji aplikacji na serwerze WWW można wywoływać ją z JavaScriptu, przekazując referencję do strony PHP, jaką chce się wywołać, w ten sam sposób jak w przypadku ASP.NET — za pomocą metody `open` obiektu `XMLHttpRequest`, jak poniżej:

```
XMLHttpRequestObject.open("POST", "response.php?value=1", "true");
```

Tak naprawdę PHP może bez żadnego problemu działać obok ASP i ASP.NET.

Rysunek 3.9.
Zakładka Filtry ISAPI



Przykład wykorzystujący Ajaksa oraz PHP

Czas przekonwertować wcześniejszy przykład tak, by zamiast ASP.NET używał on teraz PHP. Jeśli chodzi o kod po stronie klienta, dokonanie zmian jest bardzo proste. Zmienia się tylko wywołania strony ASP.NET na stronę PHP.

> ***Jeśli Czytelnik nie chce tworzyć najpierw przykładu w ASP.NET, powinien postępować zgodnie z dwoma pierwszymi punktami z części „Spróbuj sam" z ASP.NET, a później przejść do pierwszego punktu z poniższej części „Spróbuj sam".***

Następnie w PHP stosuje się ten sam wzorzec co w przypadku kodu w ASP.NET. Kod ma dwie części — odpowiedzialną za działania związane z koszykiem z zakupami (dodawanie i usuwanie przedmiotów) oraz za serializację zawartości koszyka z zakupami do postaci dokumentu XML.

spróbuj sam Przykład koszyka z zakupami w PHP

1. Należy otworzyć stronę *Catalogue1.htm* i zmienić następujący wiersz, po czym zapisać ją jako *Catalogue1PHP.htm*:

```
...
   <head>
      <script type="text/javascript" src="CartPHP.js"></script>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
   </head>
...
```

2. Należy zmienić podkreślone wiersze w pliku *Cart.js* w funkcji `AddRemoveItem`, by miały one widoczną poniżej formę, a następnie zapisać ten plik jako *CartPHP.js*:

```
if(action=="Add")
{
   xHRObject.open("GET", "ManageCart.php?action=" + action + "&book=" +
   book + "&value=" + num, true);
}
else
{
   xHRObject.open("GET", "ManageCart.php?action=" + action + "&book=" +
   book + "&value="  + num, true);
}
```

3. Teraz należy utworzyć nową stronę o nazwie *ManageCart.php* i dodać do niej następujący kod:

```
<?php session_register('Cart'); header('Content-Type: text/xml'); ?>
<?php
   $newitem = $_GET["book"];
   $action = $_GET["action"];
   if ($_SESSION["Cart"] != "")
   {
      $MDA = $_SESSION["Cart"];
      if ($action == "Add")
      {
         if ($MDA[$newitem] != "")
         {
            $value = $MDA[$newitem] + 1;
            $MDA[$newitem] = $value;
            $_SESSION["Cart"] = $MDA;
            ECHO (toXml($MDA));
         }
         else
         {
            $MDA[$newitem] = "";
            $_SESSION["Cart"] = $MDA;
            ECHO (toXml($MDA));
         }
      }
      else
      {
         $MDA= "";
         $_SESSION["Cart"] = "";
         ECHO (toXml($MDA));
      }
   }
   else
   {
      $MDA[$newitem] = "1";
      $_SESSION["Cart"] = $MDA;
      ECHO (toXml($MDA));
   }

   function toXml($MDA)
   {
      $doc = new DomDocument('1.0');
```

```
        $cart = $doc->createElement('cart');
        $cart = $doc->appendChild($cart);

        foreach ($MDA as $Item => $ItemName)
        {
           $book = $doc->createElement('book');
           $book = $cart->appendChild($book);

           $title = $doc->createElement('title');
           $title = $book->appendChild($title);
           $value = $doc->createTextNode($Item);
           $value = $title->appendChild($value);

           $quantity = $doc->createElement('quantity');
           $quantity = $book->appendChild($quantity);
           $value2 = $doc->createTextNode($ItemName);
           $value2 = $quantity->appendChild($value2);
        }

        $strXml = $doc->saveXML();
        return $strXml;
     }
?>
```

4. Należy uruchomić przykład, zaczynając od strony *Catalogue1PHP.htm*. Powinien on działać dokładnie tak samo, jak poprzednio.

Jak to działa

W kodzie po stronie klienta nie zmieniono prawie nic — poza zmianą referencji na kod PHP. W czym zatem kod w PHP jest inny? PHP nie ma na przykład wbudowanej obsługi tablic mieszających jako struktur danych. Zamiast tego należy skorzystać z wielowymiarowej tabeli, w której przechowa się zmienną `Session`. Poza tym jednak kod ten prawie dokładnie odpowiada kodowi w ASP.NET.

Ponownie trzeba zająć się czterema scenariuszami:

- Użytkownik dodaje przedmiot i przedmiot ten jest już obecny w koszyku z zakupami.
- Użytkownik dodaje nowy przedmiot, którego nie ma jeszcze w koszyku z zakupami, natomiast sam koszyk z zakupami już istnieje.
- Użytkownik usuwa przedmiot z koszyka z zakupami.
- Użytkownik dodaje przedmiot, ale koszyk jeszcze nie istnieje, dlatego trzeba go utworzyć.

Ponieważ w PHP nie jest wymagane silne typowanie zmiennych, kod ten może być nieco krótszy i prostszy od kodu w ASP.NET. Ponieważ nie jest wymagane określone zdarzenie `on_load`, kod PHP, jaki ma być wykonywany za każdym wywołaniem strony, można umieścić w jej części `head`.

Rozpoczyna się od zarejestrowania zmiennej `Session` o nazwie `Cart`, która przechowuje koszyk z zakupami. Ustawia się też `Content-Type` odpowiedzi na `text/xml`, by można było skorzystać z właściwości `responseXML`.

```
<?php session_register('Cart'); header('Content-Type: text/xml'); ?>
```

Następnie tworzy się dwie zmienne, które będą przechowywały tytuł książki oraz działanie, jakie użytkownik chce wykonać. Są one pobierane ze zbioru `$_GET`, który został dodany jako łańcuch znaków zapytania do żądania wykonanego przez JavaScript:

```
$newitem = $_GET["book"];
$action = $_GET["action"];
```

Później sprawdza się zmienną `$_Session["Cart"]`, by przekonać się, czy jest ona pusta. Jeśli nie jest pusta, tworzy się zmienną o nazwie `$MDA`, w której przechowana zostanie zmienna `Session`. Ponieważ nie używa się silnego typowania, nie trzeba się martwić o rzutowanie zmiennych PHP z jednego typu na drugi. Zazwyczaj odbywa się to automatycznie.

```
if ($_SESSION["Cart"] != "")
   {
      $MDA = $_SESSION["Cart"];
```

Dalej sprawdza się, czy zmienna `$action` zawiera słowo `Add`, które zostało na początku przekazane z JavaScriptu. Jeśli tak jest, mamy do czynienia z jednym z dwóch scenariuszy.

Pierwszy z nich to sytuacja, w której ma się już przedmiot w koszyku z zakupami i chce się dodać kolejny, przechować nową wartość w tablicy, przechować tablicę w zmiennej `Session`, a następnie dokonać serializacji tablicy, jak poniżej:

```
if ($action == "Add")
{
   if ($MDA[$newitem] != "")
   {
      $value = $MDA[$newitem] + 1;
      $MDA[$newitem] = $value;
      $_SESSION["Cart"] = $MDA;
      ECHO (toXml($MDA));
   }
```

Ta tablica działa bardzo podobnie do tablicy mieszającej z ASP.NET. Do każdego elementu można się odnieść za pomocą tytułu przechowywanej książki. Wartością indeksu tablicy jest po prostu tytuł książki. Jeśli w koszyku nie ma przedmiotów, wartość ustawia się na pustą, przechowuje tablicę w zmiennej sesyjnej i serializuje ją do postaci XML.

```
   else
   {
      $MDA[$newitem] = "";
      $_SESSION["Cart"] = $MDA;
      ECHO (toXml($MDA));
   }
}
```

W trzecim scenariuszu usuwa się przedmiot poprzez ustawienie tablicy oraz zmiennej `Session` na puste łańcuchy znaków, a następnie dokonuje się serializacji do postaci XML:

```
    else
    {
       $MDA= "";
       $_SESSION["Cart"] = "";
       ECHO (toXml($MDA));
    }
}
```

W ostatnim, czwartym scenariuszu (gdy koszyk z zakupami nie istnieje) wykorzystano ten sam kod co w pierwszym:

```
else
{
   $MDA[$newitem] = "1";
   $_SESSION["Cart"] = $MDA;
   ECHO (toXml($MDA));
}
```

Wartość ustawia się na `1`. Tworzy się zmienną `Session` o nazwie `Cart` i przypisuje się do niej tablicę. Następnie dokonuje się serializacji tablicy do postaci XML.

Choć PHP nie posiada wbudowanego zbioru metod służących specjalnie do tworzenia dokumentów XML, posiada metodę `DomDocument`. W tym przykładzie można utworzyć dokument XML jako dokument DOM, a następnie utworzyć elementy XML i dodawać je do dokumentu wraz z węzłami tekstowymi tak samo, jak w ASP.NET. Do dyspozycji jest element główny `<cart>`, wewnątrz którego tworzy się element `book` dla każdego tytułu znajdującego się w koszyku z zakupami. Element `book` zawiera elementy `<title>` oraz `<quantity>`, do których przypisuje się węzły tekstowe przechowujące dane.

```
function toXml($MDA)
{
   $doc = new DomDocument('1.0');
   $cart = $doc->createElement('cart');
   $cart = $doc->appendChild($cart);

   foreach ($MDA as $Item => $ItemName)
   {
      $book = $doc->createElement('book');
      $book = $cart->appendChild($book);

      $title = $doc->createElement('title');
      $title = $book->appendChild($title);
      $value = $doc->createTextNode($Item);
      $value = $title->appendChild($value);

      $quantity = $doc->createElement('quantity');
      $quantity = $book->appendChild($quantity);
      $value2 = $doc->createTextNode($ItemName);
      $value2 = $quantity->appendChild($value2);
   }

   $strXml = $doc->saveXML();
   return $strXml;
}
```

Funkcję `saveXML` wykorzystuje się do skompilowania dokumentu jako jednej całości. Przechowuje się go jako łańcuch znaków i zwraca do funkcji `echo`. Funkcja ta zapisuje dane do strumienia `Response` i przekazuje informacje z powrotem z serwera do skryptu po stronie klienta, w którym dokument XML zostaje wykorzystany do złożenia strony HTML.

Serwlety Javy

Ostatnia omówiona technologia po stronie serwera wykorzystuje Javę. Choć środowisko uruchomieniowe Javy może być obecne na kliencie, tym razem przedstawione zostanie wykorzystywanie Javy na serwerze z użyciem serwletów. Serwlety (ang. *servlet*) to aplikacje po stronie serwera napisane w języku programowania Java. Są wykorzystywane do tworzenia interaktywnych stron internetowych i udostępniają te same możliwości, jakich można oczekiwać od PHP czy ASP.NET (na przykład przetwarzanie formularzy, zarządzanie sesją oraz cookies, obsługa logowania).

Tak jak w przypadku PHP, choć Java jest technologią własnościową[2], jest dystrybuowana za darmo przez firmę Sun i dostępna pod adresem *http://java.sun.com*.

Jak na ironię, Ajax jest obecnie wykorzystywany do wypełniania luki, jaką Java miała wypełnić dekadę temu. Na początku Java wykorzystywała wirtualną maszynę Javy (JVM, Java Virtual Machine), by uruchamiać po stronie klienta aplikacje, które byłyby bardziej interaktywne i miałyby większe możliwości.

Akurat to zastosowanie Javy ograniczyło kilka problemów. Pierwszym z nich było to, że znacznik `<applet>`, który był dodatkiem zaakceptowanym przez Netscape (co zostało ustandaryzowane w HTML 3.2), wykorzystywanym do uruchamiania apletów Javy, szybko — w wersji 4.0 HTML — uznany został za przestarzały i odradzany; traktowany był również jako swego rodzaju aberracja. Tak naprawdę cały standard HTML 3.2 był postrzegany jako nieco samowolny.

Zaakceptowanym zamiennikiem został znacznik `<object>`, który od dłuższego czasu obecny był w przeglądarce Internet Explorer, dlatego w efekcie brakowało standardowego sposobu dodawania aplikacji w Javie. Na dodatek wirtualna maszyna Javy, która wymagana była do uruchomienia Javy, była stale uaktualniana, natomiast wersje dołączone do przeglądarek zostawały te same, o ile się ich osobno nie uaktualniło. W rzeczywistości oznaczało to, że, by uruchomić nowsze aplety Javy, często trzeba było najpierw pobrać zupełnie nową wersję wirtualnej maszyny Javy.

Ostatnim powodem niepowodzenia było to, że Java przeniosła prawie cały ciężar przetwarzania z powrotem na klienta. Aplety Javy często pobierały się bardzo powoli, a jeszcze wolniej zaczynały działać — w zależności od mocy maszyny użytkownika. Choć Java nadal jest bardzo popularnym językiem oraz środowiskiem programowania, Ajax zdecydowanie przejął jej rolę po stronie klienta.

---

[2] W listopadzie 2006 firma Sun ogłosiła, że kod źródłowy Javy zostanie opublikowany jako wolne oprogramowanie na licencji GNU GPL. Stało się tak w okresie pomiędzy listopadem 2006 a majem 2007; jedynie fragmenty pochodzące od innych podmiotów, które nie zgodziły się na opublikowanie ich na licencji GNU GPL, pozostały kodem własnościowym. Sun planuje je jednak zastąpić implementacjami alternatywnymi — *przyp. tłum.*

To, że Ajax zajął miejsce Javy po stronie klienta, nie oznacza, iż nie można z powodzeniem połączyć Ajaksa z Javą działającą na serwerze.

Przykład wykorzystujący Ajaksa oraz serwlety Javy

Wzorzec powinien już być znany. Czas przystosować przykład omawiany w niniejszym rozdziale tak, by w aplikacji opartej na Ajaksie korzystał on z serwletów Javy.

By tworzyć i uruchamiać serwlety, potrzebne będą następujące elementy:

- J2EE SDK, wersja 2 (*http://java.sun.com/products/servlet/download.html*) — to aktualna wersja języka programowania Java,
- Tomcat@Jakarta (*http://tomcat.apache.org/*) — serwer, na którym można uruchamiać serwlety.

Serwlety wymagają serwera z obsługą Javy wraz z silnikiem obsługującym serwlety. Należy zainstalować J2EE SDK zgodnie z załączonymi instrukcjami i uruchomić serwer aplikacji Javy. Serwer ten najczęściej działa pod adresami *http://localhost:8080*, *http://localhost:8081* bądź *http://localhost:4848*. Następnie należy zainstalować serwer Tomcat i uruchomić usługę, na której jest on oparty, zgodnie ze wskazówkami dostarczonymi przy instalacji.

Kiedy aplikacja zostanie poprawnie zainstalowana na serwerze WWW, można ją wywołać z JavaScriptu w podobny (choć nieco odmienny) sposób jak poprzednio. Serwlety muszą być skompilowane, zanim będzie się można do nich odnosić, dlatego należy skompilować klasy Javy w pliki *.class*. Następnie należy się odnieść do pliku *.class* za pomocą metody `open` obiektu `XMLHttpRequest`, jak poniżej:

```
XMLHttpRequestObject.open("POST", "response?value=1", "true");
```

I znów przykład ten wymaga minimalnej interwencji w przypadku kodu po stronie klienta. Zmieniony zostanie jedynie kod po stronie serwera, a ponieważ w przykładzie tym wykorzystano serwer Tomcat, konieczna będzie również zmiana lokalizacji plików aplikacji. Tomcat wymaga także wykonania kilku czynności konfiguracyjnych, by serwlet działał.

spróbuj sam Przykład koszyka z zakupami w Javie

1. Należy umieścić wszystkie pliki w folderze *C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\ROOT* (lub w innym folderze, w którym zainstalowano serwer Tomcat). Należy otworzyć plik *Catalogue1.htm*, zmienić poniższy wiersz i zapisać plik jako *Catalogue1Java.htm*:

```
...
   <head>
      <script type="text/javascript" src="CartJava.js"></script>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
   </head>
...
```

2. Należy zmienić poniższe podkreślone wiersze w pliku *Cart.js* w funkcji `AddRemoveItem`, by miały one widoczną poniżej formę, a następnie zapisać ten plik jako *CartJava.js*:

```
        if(action=="Add")
        {
           xHRObject.open("GET", "ManageCartServlet?action=" + action + "&book=" +
           book + "&value=" + num, true);
        }
        else
        {
           xHRObject.open("GET", "ManageCartServlet?action=" + action + "&book=" +
           book + "&value="  + num, true);
        }
```

Należy również zmienić poniższe wiersze:

```
            if (window.ActiveXObject)
            {
               spantag.innerHTML += " " +header[0].firstChild.text;
               spantag.innerHTML += " " + header[0].lastChild.text + " " + "<a
href='#' onclick='AddRemoveItem(\"Remove\");'>Usuń przedmiot</a>";
            }
            else
            {
               spantag.innerHTML += " " +header[0].childNodes[1].textContent;
               spantag.innerHTML += " " + header[0].childNodes[3].textContent +
               " " + "<a href='#' onclick='AddRemoveItem(\"Remove\");'>Usuń
               przedmiot</a>";
            }
```

3. Następnie należy utworzyć plik tekstowy o nazwie *ManageCartServlet.java* i dodać do niego następujący kod:

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.ServletException;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Iterator;

public class ManageCartServlet extends HttpServlet
{
   public void doGet(HttpServletRequest req, HttpServletResponse res)
         throws IOException, ServletException
   {
      String newitem = req.getParameter("book");
      String action = req.getParameter("action");
      Hashtable ht = new Hashtable();
      HttpSession session = req.getSession(true);
      if (session.getAttribute("cart") != null)
      {
         ht = (Hashtable) session.getAttribute("cart");
         if ("Add".equals(action))
         {
            if (ht.containsKey(newitem))
            {
               int value = 1;
               if (ht.containsKey(newitem))
```

```
                {
                   Integer currentQuantity = (Integer) ht.get(newitem);
                   value += currentQuantity.intValue();
                }
                ht.put(newitem, new Integer(value));
                session.setAttribute("cart", ht);
                String cartXml = toXml(ht);
                res.setContentType("text/xml");
                res.getWriter().write(cartXml);
             } else
             {
                ht.put(newitem, 1);
                session.setAttribute("cart", ht);
                String cartXml = toXml(ht);
                res.setContentType("text/xml");
                res.getWriter().write(cartXml);
             }
          } else
          {
             ht.remove(newitem);
             session.setAttribute("cart", null);
             String cartXml = toXml(ht);
             res.setContentType("text/xml");
             res.getWriter().write(cartXml);
             ;
          }
       } else
       {
          ht.put(newitem, 1);
          session.setAttribute("cart", ht);
          String cartXml = toXml(ht);
          res.setContentType("text/xml");
          res.getWriter().write(cartXml);
       }
    }

    public String toXml(Hashtable ht)
    {
       StringBuffer xmlDoc = new StringBuffer();
       xmlDoc.append("<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=
       \"yes\"?>\n");
       xmlDoc.append("<cart>\n");
       for (Iterator<String> x = ht.keySet().iterator(); x.hasNext();)
       {
          String item = x.next();
          int Quantity = ((Integer) ht.get(item)).intValue();
          xmlDoc.append("<book>\n");
          xmlDoc.append("<title>");
          xmlDoc.append(item);
          xmlDoc.append("</title>\n");
          xmlDoc.append("<quantity>");
          xmlDoc.append(Quantity);
          xmlDoc.append("</quantity>\n");
          xmlDoc.append("</book>\n");
       }
       xmlDoc.append("</cart>\n");
       return xmlDoc.toString();
    }
}
```

4. Należy skompilować aplikację do pliku *.class*. JDK z firmy Sun zawiera kompilator wiersza poleceń `javac`. Przy kompilacji niezbędne będzie również dołączenie API serwletu w następujący sposób (warto zwrócić uwagę na to, że adres URL może się różnić w zależności od instalacji serwera Tomcat). Dla wygody plik *ManageCartServlet.java* umieszczony został w tym samym folderze co kompilator *javac.exe* z poniższego polecenia:

```
javac -classpath ".;C:\Program Files\Apache Software Foundation\Tomcat 5.5\
common\lib\servlet-api.jar" ManageCartServlet.java
```

5. W domyślnej konfiguracja serwera Tomcat skompilowane serwlety umieszczane są w folderze *webapps\ROOT\WEB-INF\classes* katalogu instalacyjnego Tomcata. Należy umieścić plik *ManageCartServlet.class* w tym katalogu. Jeśli uruchamia się serwer po raz pierwszy, należy najpierw utworzyć odpowiedni folder.

6. Należy utworzyć następujący plik *web.xml* i zapisać go w folderze *WEB-INF* znajdującym się w folderze *ROOT*:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">

   <display-name>Aplikacja ManageCartServlet</display-name>
   <description>Testowanie Ajaksa</description>

   <servlet>
      <servlet-name>ManageCartServlet</servlet-name>
      <servlet-class>ManageCartServlet</servlet-class>
   </servlet>

   <servlet-mapping>
      <servlet-name>ManageCartServlet</servlet-name>
      <url-pattern>/ManageCartServlet</url-pattern>
   </servlet-mapping>

</web-app>
```

7. Teraz należy uruchomić przykład raz jeszcze, otwierając stronę *http://localhost:8080/Catalogue1Java.htm*, jednak tym razem z użyciem Javy. Przykład będzie działał dokładnie tak samo, jak w przypadku ASP.NET oraz PHP.

Jak to działa

Przykład ten w pełni odzwierciedla dwa poprzednie, choć wymaga dodatkowej konfiguracji, by zaczął działać. Ponownie wykorzystuje się dwie metody, które wykonują pożądane zadania.

Metoda `doGet` przypomina metodę `OnLoad` z ASP.NET, ponieważ jest ona wywoływana za każdym razem, gdy uruchamiana jest strona. Jeśli Czytelnik nie jest zaznajomiony z ASP.NET, wystarczy wiedza, że jest to metoda wywoływana za każdym razem, gdy strona rozpoczyna działanie. Zazwyczaj w pliku PHP jest to kod, który nie został ujęty wewnątrz funkcji.

Czas powrócić do czterech scenariuszy. Nie ma sensu ich powtarzać kolejny raz ani też dokładnie analizować podobnego kodu. Zamiast tego warto zwrócić uwagę na najistotniejsze różnice.

Ponownie można skorzystać z tablicy mieszającej do przechowania informacji o koszyku z zakupami, a tablica ta mieści się w zmiennej `Session`. Z punktu widzenia semantyki obsługa sesji w Javie jest nieco bardziej rozwlekła niż w ASP.NET oraz PHP, jednak w dużej mierze działa w ten sam sposób. Ustawia się wartości i przechowuje je w tablicy mieszającej. Wykorzystuje się klucz tablicy mieszającej do przechowania tytułu książki. Ustawia się zawartość zmiennej `Session`, `Content-Type` odpowiedzi, a także zapisuje zserializowaną (za pomocą metody `toXML`) tablicę mieszającą w niemalże identyczny sposób jak w przykładach opartych na ASP.NET oraz PHP.

Metoda `toXML` jest nieco bardziej prymitywna od jej odpowiedników w ASP.NET oraz PHP i by przykład ten pozostał prosty, utworzono obiekt `StringBuffer`, po czym dynamicznie utworzono łańcuch znaków z dokumentem XML:

```
    public String toXml(Hashtable ht)
    {
       StringBuffer xmlDoc = new StringBuffer();
       xmlDoc.append("<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"yes\"?>\n");
       xmlDoc.append("<cart>\n");

       for (Iterator<String> x = ht.keySet().iterator(); x.hasNext();)
       {
          String item = x.next();
          int Quantity = ((Integer) ht.get(item)).intValue();
          xmlDoc.append("<book>\n");
          xmlDoc.append("<title>");
          xmlDoc.append(item);
          xmlDoc.append("</title>\n");
          xmlDoc.append("<quantity>");
          xmlDoc.append(Quantity);
          xmlDoc.append("</quantity>\n");
          xmlDoc.append("</book>\n");
       }
       xmlDoc.append("</cart>\n");
       return xmlDoc.toString();
    }
}
```

Dane te są zwracane do metody `doGet()` wraz ze zserializowanym dokumentem XML, a stamtąd z powrotem do klienta. Ponieważ ustawiono `Content-Type` odpowiedzi na `text/xml`, dokument XML zostanie poprawnie przetworzony przez obiekt `XMLHttpRequest` pomimo tego, iż na serwerze został on utworzony jako łańcuch znaków.

Którą technologię powinno się wykorzystywać?

Autorzy książki mają nadzieję, że Czytelnik zapoznał się z najpopularniejszymi technologiami po stronie serwera. ASP.NET oraz PHP działają na serwerze IIS. PHP można również uruchomić na serwerze Apache, natomiast serwlety Javy wymagają własnego serwera aplikacji. Żaden z przykładów nie różni się znacząco od pozostałych, a najbardziej skomplikowanymi ich częściami są chyba instalacja technologii po stronie serwera i wykorzystanie jej do osiągnięcia zamierzonego celu.

Celem niniejszego omówienia nie jest ocena zalet oraz wad wymienionych technologii. Zazwyczaj to konieczność napędza programistów, więc na ogół albo klient nakazuje użycie określonej technologii, albo też autor aplikacji zna jedną z nich na tyle dobrze, że to na niej oprze swoją aplikację.

Oczywiście czasami jedna z technologii lepiej od pozostałych nadaje się dla konkretnej implementacji. ASP.NET posiada o wiele bardziej rozbudowaną i skomplikowaną obsługę stanów od PHP, a także zawiera alternatywny sposób zarządzania stanem, który nie wiąże się z wykorzystywaniem cookies. PHP oraz Java działają na większej liczbie platform niż ASP.NET, jednak ASP.NET staje się coraz bardziej wszechobecny. Jeśli chodzi o odpowiedź na pytanie, czy jedna z tych technologii działa lepiej z Ajaksem, czy też lepiej do Ajaksa pasuje, odpowiedź może być tylko jedna — nie. Tak naprawdę lepiej spojrzeć na to z drugiej strony — Ajax wypełnia lukę, którą należało wypełnić w każdej z tych technologii.

Wybór należy zatem do programisty i nie powinien on odrzucać Ajaksa tylko dlatego, że jest przywiązany do określonej platformy programistycznej czy technologii. Ajax działa równie dobrze na każdej z nich.

Podsumowanie

Niniejszy rozdział był krótkim spojrzeniem na rolę serwera w aplikacjach opartych na Ajaksie. Jeśli Czytelnik zainteresował się Ajaksem i oczekiwał, że w technologii tej nie ma się do czynienia z serwerem, może się czuć nieco zawiedziony. W przypadku większości aplikacji nie da się tego prawdopodobnie obejść. Rozpoczęto od zbadania roli serwera w tradycyjnym cyklu żądanie-odpowiedź w programowaniu webowym, a następnie porównano ją z wykonywaniem zapytań do serwera w Ajaksie oraz JavaScripcie. Następnie rozszerzono omówienie każdego z kroków tego cyklu.

Później krótko omówiono każdą z trzech wybranych technologii po stronie serwera. Utworzono przykład z koszykiem z zakupami oraz dopasowano do niego stronę po stronie serwera napisaną w ASP.NET, PHP oraz Javie, by zobaczyć, jak niewiele różnic występuje pomiędzy tymi technologiami. Niewiele miejsca poświęcono operacjom po stronie klienta, jednak wspomniano nieco o tym, w jaki sposób obiekt `XMLHttpRequest` wykorzystywany jest do inicjowania żądań i otrzymywania danych.

W rozdziale 4. kwestie te omówione zostaną o wiele bardziej szczegółowo wraz z metodami wykorzystywanymi w Ajaksie do tworzenia żądania oraz obsługiwania jego wyników.

Ćwiczenia

Sugerowane rozwiązania poniższych zadań znajdują się w dodatku A.

1. Należy dodać numer ISBN do dokumentu XML. Dlaczego może to być użyteczne?
2. Należy ulepszyć koszyk z zakupami, tak by numer ISBN był teraz w nim widoczny.