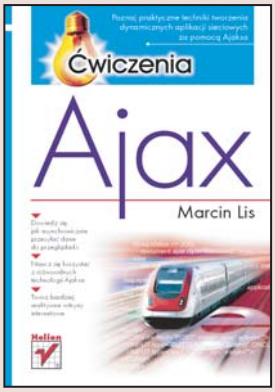
Ajax. Ćwiczenia

Autor: Marcin Lis
ISBN: 83-246-0686-6
Format: A5, stron: 168
Przykłady na ftp: 32 kB



Ajax to nowe podejście do tworzenia witryn internetowych bazujące na różnorodnych technologiach, takich jak HTML i JavaScript, obiekty XMLHttpRequest, style CSS, model DOM czy XML. Charakterystyczne dla stron budowanych zgodnie z tym podejściem jest to, że nie wymagają one przeładowywania. Dzięki temu, że przesyłane są tylko niezbędne dane, takie witryny internetowe działają w sposób zbliżony do standardowych aplikacji. Daje to wiele nowych możliwości, dlatego Ajax zasłużenie szybko zyskał dużą popularność.

„Ajax. Ćwiczenia" to zbiór praktycznych ćwiczeń pokazujących, jak za pomocą technologii związanych z Ajaksem oraz języka PHP wykonywać operacje pozwalające tworzyć bardziej reaktywne witryny. Wykonując kolejne ćwiczenia, nauczysz się między innymi wysyłać i pobierać dane za pomocą obiektów XMLHttpRequest oraz dynamicznie aktualizować strony. Dowiesz się, w jaki sposób Ajax współpracuje ze skryptami PHP oraz jak obsługiwać dane XML. Poznasz także wiele różnych technik pozwalających na efektywne przesyłanie informacji między serwerem a przeglądarką w synchroniczny i asynchroniczny sposób. Dzięki lekturze tej książki nauczysz się tworzyć bogatsze i bardziej reaktywne witryny internetowe.

- Przesyłanie danych za pomocą obiektów XMLHttpRequest
- Współpraca Ajaksa z PHP
- Obsługa danych XML
- Przesyłanie danych w różnych formatach
- Generowanie kodu po stronie serwera
- Obiekty JSON

Tchnij więcej życia w swoje witryny internetowe

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

Spis treści

AJAX i XML

Krótko o XML

Skrót AJAX oznacza *Asynchronous Javascript and XML* — z samej nazwy wynika więc, że technika ta jest związana z językiem XML. Co prawda, jak pokazały poprzednie rozdziały, stosowanie XML-a wcale nie jest konieczne do tworzenia aplikacji AJAX-a, warto jednak wiedzieć, jak wygląda współpraca między XML-em i AJAX-em oraz jak się tworzy, pobiera i przetwarza tego typu dane. Takim właśnie zagadnieniom poświęcony jest czwarty rozdział.

XML to *EXtensible Markup Language*, czyli rozszerzalny język znaczników. Jest to rozwijany przez organizację W3C (*http://www.w3.org*) niezależny od platformy systemowej język opisu danych. Faktycznie XML służy do wyprowadzania innych języków, tzw. aplikacji XML. Pozwala na definiowanie struktury danych, struktury dokumentów, zupełnie niezależnej od sposobu ich prezentacji, jak jest np. w HTML. Nie ma w nim też narzuconego z góry zestawu znaczników. I to jest jego największa siła. Dzięki tym cechom umożliwia łatwą wymianę danych oraz bezproblemowe ich przetwarzanie.

Dokument XML składa się z nagłówka oraz zbioru znaczników definiujących elementy dokumentu. Każdy element musi mieć znacznik otwierający i zamykający, np.:

```
<element1>
dane
</element1>
```

chyba że jest elementem pustym (czyli niezawierającym danych). W tym ostatnim przypadku znacznik będzie miał postać:

```
<znacznik />
```

W nagłówku należy natomiast określić wersję języka. W dalszych ćwiczeniach będziemy stosować nagłówek w postaci:

```
<?xml version="1.0" ?>
```

Elementy dokumentu XML mogą być zagnieżdżane (podobnie jak znaczniki HTML), np.:

```
<element1>
  <element2>
  dane
  </element2>
</element1>
```

Mogą także posiadać atrybuty, choć w niniejszej publikacji nie będziemy ich stosować. Jak może wyglądać prosty dokument XML? Na przykład następująco:

```
<?xml version="1.0" ?>
<dokument>
To jest pierwszy dokument!
</dokument>
```

Mamy tu znacznik otwierający `<dokument>` i zamykający `</dokument>`. Te znaczniki definiują element XML o nazwie `dokument`, natomiast tekst znajdujący się między nimi to zawartość tego elementu. Przy czym `<dokument>` bynajmniej nie jest predefiniowanym elementem języka — jest to nasz własny znacznik, który sobie wymyśliliśmy. Równie dobrze można go nazwać np. `napis`:

```
<?xml version="1.0" ?>
<napis>
To jest pierwszy dokument!
</napis>
```

Te wiadomości wystarczą już do wykonania ćwiczeń znajdujących się na kolejnych stronach.

Reprezentacja dokumentów XML

W poprzednich rozdziałach zaprezentowano wiele ćwiczeń pokazujących, w jaki sposób odbierać dane tekstowe za pomocą obiektu `XMLHttpRequest`. Dane te były zarówno statyczne, jak i generowane dynamicznie, jednak dostęp do nich zawsze był uzyskiwany przez odwołanie się do właściwości `responseText`. Jeśli zajrzymy teraz do tabeli 1.1 z rozdziału 1., znajdziemy w niej również właściwość `responseXML`. Właśnie ona zawiera dane w formacie XML, o ile oczywiście w takiej formie zostały przesłane z serwera. W rzeczywistości właściwość ta zawiera obiekt posiadający właściwości pozwalające na dostęp do danych zawartych w pobranym dokumencie XML. Typowe właściwości zostały przedstawione w tabeli 4.1.

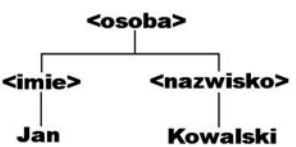
Tabela 4.1. *Właściwości pozwalające na dostęp do struktury dokumentu XML*

Nazwa	Opis
`attributes`	Atrybuty węzła
`childNodes`	Tablica węzłów potomnych
`documentElement`	Główny element dokumentu
`firstChild`	Pierwszy węzeł potomny
`lastChild`	Ostatni węzeł potomny
`localName`	Lokalna nazwa węzła
`name`	Nazwa węzła
`nextSibling`	Kolejny węzeł sąsiadujący
`nodeName`	Nazwa węzła
`nodeType`	Typ węzła
`nodeValue`	Wartość węzła
`previousSibling`	Poprzedni węzeł sąsiadujący

Aby zrozumieć znacznie tych właściwości, trzeba wiedzieć, że dokument XML w obiekcie wskazanym przez właściwość `responseXML` jest traktowany jako struktura drzewiasta, w której poszczególne elementy dokumentu są węzłami. Załóżmy, że mamy do dyspozycji dokument XML w postaci:

```
<?xml version="1.0" ?>
<osoba>
  <imie>
    Jan
  </imie>
  <nazwisko>
    Kowalski
  </nazwisko>
</osoba>
```

Będzie on reprezentowany jako struktura widoczna na rysunku 4.1.



Rysunek 4.1. *Drzewiasta struktura dokumentu XML*

Jak widać, taki dokument posiada węzeł (ang. *node*) odpowiadający elementowi `<osoba>`, zaś węzeł ten posiada dwa podwęzły (ang. *subnodes*), czyli węzły potomne (ang. *child nodes*): `<imie>` i `<nazwisko>`. Każdy z węzłów potomnych posiada dodatkowo podwęzeł przechowujący tekstowe dane zdefiniowane w elementach `<imie>` i `<nazwisko>`. Wynika z tego, że:

- węzeł `<osoba>` jest węzłem głównym,
- właściwość `firstChild` węzła `<osoba>` to węzeł `<imie>`,
- właściwość `lastChild` węzła `<osoba>` to węzeł `<nazwisko>`,
- właściwość `firstChild` węzła `<imie>` to węzeł tekstowy zawierający ciąg znaków `Jan`,
- właściwość `firstChild` węzła `<nazwisko>` to węzeł tekstowy zawierający ciąg znaków `Kowalski`,
- właściwość `nextSibling` węzła `<imie>` to węzeł `<nazwisko>`,
- właściwość `previousSibling` węzła `<nazwisko>` to węzeł `<imie>`
- itd.

Tak więc, jeśli odbierzemy dane i dokonamy przypisania:

```
var xml = XMLHttpRequestObject.responseXML;
```

to aby dostać się do węzła `<osoba>`, napiszemy:

```
xml.documentElement
```

Aby dostać się do węzła <imie>, wpiszemy:

```
xml.documentElement.firstChild
```

bowiem węzeł <imie> jest pierwszym węzłem potomnym (firstChild) węzła <osoba>.

Aby dostać się do węzła <nazwisko>, napiszemy:

```
xml.documentElement.lastChild
```

bowiem węzeł <nazwisko> jest ostatnim węzłem potomnym (lastChild) węzła <osoba>.

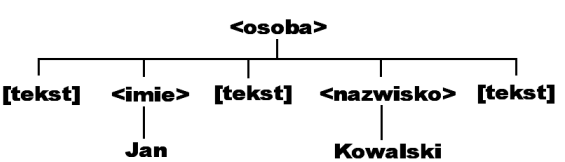
Aby dostać się do węzła tekstowego zawierającego imię, napiszemy:

```
xml.documentElement.firstChild.firstChild
```

bowiem węzeł ten jest pierwszym (i zarazem jedynym) węzłem potomnym węzła <imie>.

Można by przytoczyć jeszcze kilka innych konstrukcji (np. wykorzystujących węzły sąsiadujące), nie ma chyba jednak takiej potrzeby — zasady dostępu z pewnością są już jasne.

Niestety powyższy opis sprawdzi się w pełni tylko w przypadku przeglądarki Internet Explorer. Przeglądarki oparte na module Gecko (np. Firefox), ale również np. Opera nieco inaczej interpretują dane XML. Otóż białe znaki (a więc znaki końca linii, tabulacji, spacje itp.) znajdujące się w treści dokumentu pomiędzy znacznikami są traktowane jako węzły tekstowe. Stąd przedstawiony wyżej dokument — jako obiekt XML w przeglądarce — będzie miał postać przedstawioną na rysunku 4.2.



Rysunek 4.2. *Struktura dokumentu XML w przeglądarkach Firefox, Opera itp.*

Tak więc w tym przypadku węzeł <osoba> również jest węzłem głównym, ale pierwszym węzłem potomnym jest węzeł tekstowy zawierający znak końca linii i spacje. Dopiero pierwszym sąsiadem tego węzła jest węzeł <imie>. Jeśli zatem odbierzemy dane i dokonamy przypisania:

```
var xml = XMLHttpRequestObject.responseXML;
```

to aby dostać się do węzła <osoba>, napiszemy:

```
xml.documentElement
```

Aby dostać się do węzła <imie>, wpiszemy:

```
xml.documentElement.firstChild.nextSibling
```

bowiem węzeł <imie> jest kolejnym węzłem sąsiadującym (nextSibling) z pierwszym węzłem potomnym (firstChild) węzła <osoba>.

Aby dostać się do węzła <nazwisko>, napiszemy:

```
xml.documentElement.lastChild.previousSibling
```

bowiem węzeł <nazwisko> jest przedostatnim węzłem potomnym (czyli wcześniejszym węzłem sąsiadującym ostatniego węzła potomnego) węzła <osoba>.

Aby dostać się do węzła tekstowego zawierającego imię, napiszemy:

```
xml.documentElement.firstChild.nextSibling.firstChild
```

bowiem węzeł ten jest pierwszym (i zarazem jedynym) węzłem potomnym węzła <imie>.

Jak widać, czasami niezbędne może okazać się pisanie dwóch wersji kodu, obsługujących dwa przedstawione sposoby reprezentacji danych. Jednak w wielu przypadkach uda nam się obejść te ograniczenia i utworzyć kod działający poprawnie w obu rodzajach przeglądarek. Pomocna może być w tym właściwość nodeType każdego węzła, pozwalająca określić jego typ. Typy węzłów i odpowiadające im wartości zostały przedstawione w tabeli 4.2. Ich znajomość przyda się przy wykonywaniu jednego z ćwiczeń przedstawionych w kolejnym podrozdziale.

Odbieranie danych XML

Dwa poprzednie podrozdziały zawierały wiele informacji na temat XML-a i sposobu reprezentacji dokumentów tego typu w przeglądarkach. Czas wykorzystać tę wiedzę i wykonać pierwszy przykład, który pokaże, jak za pomocą AJAX-a pobrać dane znajdujące się w pliku tekstowym zawierającym dokument XML oraz wyświetlić je na ekranie przeglądarki.

Tabela 4.2. *Typy węzłów dokumentu XML*

Wartość	Typ węzła
1	Element
2	Atrybut
3	Węzeł tekstowy
4	Sekcja CDATA
5	Referencja do encji
6	Węzeł encji
7	Instrukcje przetwarzania XML
8	Komentarz XML
9	Węzeł dokumentu XML
10	Definicja typu dokumentu
11	Część dokumentu XML
12	Notacja XML

ĆWICZENIE

4.1 Pobieranie danych XML

Napisz skrypt pobierający z serwera dane w formacie XML i wyświetlający je na ekranie.

Najpierw należy przygotować plik w formacie XML. Będzie on miał bardzo prostą postać:

```
<?xml version="1.0" ?>
<imie>
  Anna
</imie>
```

Cała zawartość takiego dokumentu to jeden element (określony przez znacznik `<imie>`) zawierający dane o imieniu Anna. Przedstawioną treść należy zapisać w pliku *index.xml*.

Jak pobrać w tle i przetworzyć takie dane, pokazuje skrypt widoczny na poniższym listingu:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<title>Ajax</title>
<script type="text/javascript">
var XMLHttpRequestObject = false;

if(window.XMLHttpRequest){
  XMLHttpRequestObject = new XMLHttpRequest();
}
else if(window.ActiveXObject){
  XMLHttpRequestObject = new ActiveXObject("Microsoft.XMLHTTP");
}

function pobierzDane(dest)
{
  if(XMLHttpRequestObject){
    var div = document.getElementById(dest);

    var url = "http://localhost/index.xml";
    XMLHttpRequestObject.open("GET", url);

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
          XMLHttpRequestObject.status == 200){
        var xml = XMLHttpRequestObject.responseXML;
        var imie = xml.documentElement.firstChild.nodeValue;
        var str = "Imię odczytane z dokumentu XML to: " + imie;
        div.innerHTML = str;
      }
    }
    XMLHttpRequestObject.send(null);
  }
}
</script>
</head>
<body>
<input type="button" value="Kliknij tu"
       onclick="pobierzDane('dataDiv');" />
<hr>
</div>
<div id="dataDiv">
Ten tekst zostanie zmieniony.
</div>
</body>
</html>
```

Na stronie znajduje się przycisk, któremu została przypisana procedura obsługi zdarzenia `onclick` w postaci funkcji `pobierzDane` oraz warstwa o identyfikatorze `dataDiv` służąca do wyświetlania danych. Funkcja `pobierzDane` przyjmuje jeden argument określający identyfikator warstwy, na której mają pojawić się wyniki jej działania. W jej wnętrzu na początku wykonywane są typowe operacje niezbędne do transmisji i przetwarzania danych, czyli pobranie obiektu warstwy wskazywanej przez argument `dest`:

```
var div = document.getElementById(dest);
```

podanie adresu wskazującego plik XML, który ma zostać pobrany:

```
var url = "http://localhost/index.xml";
```

oraz zainicjowanie żądania za pomocą metody `open`:

```
XMLHttpRequestObject.open("GET", url);
```

Dane tradycyjnie odczytywane są za pomocą funkcji anonimowej przypisanej właściwości `onreadystatechange` obiektu `XMLHttpRequestObject`. W przeciwieństwie jednak do przykładów prezentowanych we wcześniejszych rozdziałach, odczyt następuje przez odwołanie się do właściwości `responseXML`:

```
var xml = XMLHttpRequestObject.responseXML;
```

Po wykonaniu takiej instrukcji zmienna `xml` będzie zawierała obiekt odzwierciedlający strukturę dokumentu XML. Ponieważ nasz dokument ma bardzo prostą strukturę, gdzie elementem głównym jest znacznik `<imie>`, to dostęp do tego znacznika osiągamy przez odwołanie się do właściwości `documentElement`, natomiast dostęp do węzła tekstowego zawierającego imię — przez odwołanie się do właściwości `firstChild.nodeValue`. Jest to realizowanie przez instrukcję:

```
var imie = xml.documentElement.firstChild.nodeValue;
```

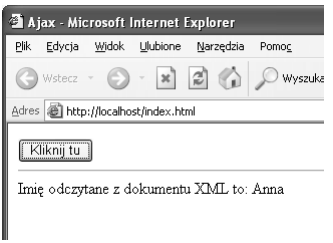
Odczytana wartość węzła tekstowego jest następnie wykorzystywana do skonstruowania napisu, który pojawi się na stronie. Jest on zapisywany w zmiennej `str`:

```
var str = "Imię odczytane z dokumentu XML to: " + imie;
```

Z kolei wartość zmiennej `str` jest przypisywana właściwości `innerHTML` warstwy wskazywanej przez obiekt `div`:

```
div.innerHTML = str;
```

Ostatecznie po kliknięciu przycisku zobaczmy widok przedstawiony na rysunku 4.3.



Rysunek 4.3. *Dane pobrane z dokumentu XML zostały wyświetlone na stronie*

Ćwiczenie 4.1 pokazało, w jaki sposób odebrać dane w formacie XML i wyświetlić je na witrynie. Struktura dokumentu była jednak bardzo prosta, nie występowały więc żadne problemy związane z różnymi sposobami reprezentacji danych stosowanych w przeglądarkach, a opisanych w poprzednim podrozdziale. Spróbujmy zatem poprawnie przetworzyć nieco bardziej złożony dokument zawierający zagnieżdżone znaczniki.

ĆWICZENIE

4.2 Przetwarzanie dokumentu XML

Napisz skrypt przetwarzający dane z dokumentu XML zawierającego kilka zagnieżdżonych znaczników.

Dokument XML, który wykorzystamy w tym ćwiczeniu (należy go zapisać w pliku *index.xml*), będzie miał postać:

```
<?xml version="1.0" ?>
<imiona>
  <imie>
    Anna
  </imie>
  <imie>
    Magdalena
  </imie>
  <imie>
    Jolanta
  </imie>
</imiona>
```

Nie jest on bardzo skomplikowany, jest to jednak struktura drzewiasta, bardziej złożona niż w poprzednim ćwiczeniu. Widzimy, że elementem głównym jest znacznik `<imiona>`. Wewnątrz niego za pomocą znaczników `<imie>` zostały zdefiniowane trzy kolejne elementy, a każdy z tych elementów zawiera dane tekstowe. Możemy być zatem pewni, że Inter-

net Explorer zinterpretuje te dane inaczej niż Firefox czy Opera. Pierwszym pomysłem, który się nasuwa, jest rozpoznanie typu przeglądarki (co i tak wykonujemy podczas tworzenia obiektu `XMLHttpRequest`) oraz napisanie dwóch różnych wersji kodu. W tym przypadku nie musimy jednak tak postępować. Zamiast tego wystarczy bowiem odpowiednio wykorzystać właściwości `childNodes` i `nodeType`, tak jak zostało to wykonane w poniższym skrypcie:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<title>Ajax</title>
<script type="text/javascript">
var XMLHttpRequestObject = false;

if(window.XMLHttpRequest){
  XMLHttpRequestObject = new XMLHttpRequest();
}
else if(window.ActiveXObject){
  XMLHttpRequestObject = new ActiveXObject("Microsoft.XMLHTTP");
}

function przetwarzajXML(xml, dest)
{
  var div = document.getElementById(dest);
  var imiona = xml.documentElement.childNodes;
  var str = "Imiona odczytane z danych XML: <br />";
  for(var i = 0; i < imiona.length; i++){
    if(imiona[i].nodeType == 1){
      str += imiona[i].firstChild.nodeValue + "<br />";
    }
  }
  div.innerHTML = str;
}

function pobierzDane(dest)
{
  if(XMLHttpRequestObject){
    var url = "http://localhost/index.xml";
    XMLHttpRequestObject.open("GET", url);

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
          XMLHttpRequestObject.status == 200){
        var xml = XMLHttpRequestObject.responseXML;
```

```
            przetwarzajXML(xml, dest);
          }
        }
        XMLHttpRequestObject.send(null);
      }
    }
    </script>
    </head>
    <body>
    <input type="button" value="Kliknij tu"
           onclick="pobierzDane('dataDiv');" />
    <hr>
    </div>
    <div id="dataDiv">
    Ten tekst zostanie zmieniony.
    </div>
    </body>
    </html>
```

Przycisk oraz warstwa z danymi zostały zdefiniowane w taki sam sposób jak w poprzednim ćwiczeniu. Funkcja `pobierzDane` ma również podobną strukturę. Pewnym zmianom uległa natomiast funkcja anonimowa przypisana właściwości `onreadystatechange` obiektu `XMLHttpRequestObject`. Najpierw pobiera ona obiekt XML będący reprezentacją danych przesłanych z serwera:

```
var xml = XMLHttpRequestObject.responseXML;
```

a następnie przekazuje go w postaci argumentu funkcji `przetwarzajXML` zajmującej się przetwarzaniem i interpretacją danych:

```
przetwarzajXML(xml, dest);
```

Drugim argumentem jest identyfikator warstwy, na której ma się pojawić tekst wynikowy.

Funkcja `przetwarzajXML` najpierw pobiera obiekt warstwy (i przypisuje go zmiennej `div`):

```
var div = document.getElementById(dest);
```

a następnie tablicę zawierającą węzły potomne węzła głównego (jest ona zapisana we właściwości `childNodes`):

```
var imiona = xml.documentElement.childNodes;
```

Ponieważ węzłem głównym jest w tym przypadku `<imiona>`, tablica ta będzie zawierała elementy zdefiniowane za pomocą znaczników `<imie>` oraz — w przypadku przeglądarek innych niż Internet Explorer — węzły tekstowe. Pozostaje więc znaleźć sposób na odróżnienie ich

od siebie i pobranie danych tylko węzłów typu <imie>. Ta czynność wykonywana jest w pętli for:

```
for(var i = 0; i < imiona.length; i++){
```

Przechodzi ona wszystkie elementy tablicy imiona, począwszy od elementu pierwszego, o indeksie 0, aż do ostatniego, o indeksie imiona.length - 1 (o jeden mniejszym niż całkowity rozmiar tablicy zapisany we właściwości lenght).

We wnętrzu pętli za pomocą instrukcji warunkowej if sprawdzany jest typ węzła zapisany we właściwości nodeType każdego z nich. W przypadku naszego dokumentu będziemy mieli do czynienia z dwoma typami węzłów:

- typem 1 — definiującym element XML (czyli określonym za pomocą znaczników <imie>),
- typem 3 — tekstowym.

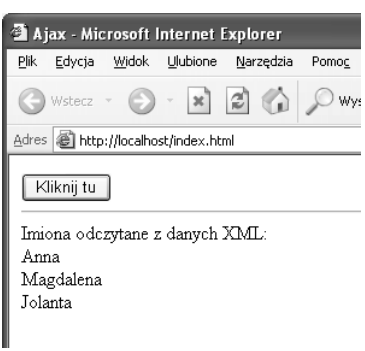
Tak więc odwołanie do imion ma sens tylko wtedy, kiedy aktualnie przetwarzany węzeł ma typ równy 1:

```
if(imiona[i].nodeType == 1){
```

W takim przypadku, ponieważ mamy wtedy do czynienia z elementem <imie>, trzeba się odwołać do pierwszego (i jedynego) węzła potomnego zawierającego dane konkretnego imienia. Dane te są dopisywane do zmiennej str:

```
str += imiona[i].firstChild.nodeValue + "<br />";
```

Dzięki temu po zakończeniu pętli mogą być one wyświetlone na dolnej warstwie, tak jak jest to widoczne na rysunku 4.4.



Rysunek 4.4. *Imiona odczytane z dokumentu XML*

Oprócz przedstawionego wyżej sposobu pobierania danych, niezależnego do przeglądarki, jaka jest do tego wykorzystywana, istnieje także inny. Otóż zamiast przeglądać „ręcznie" strukturę dokumentu, można wykorzystać metodę `getElementsByTagName`, która pobierze wszystkie węzły o zadanej nazwie i zwróci je w postaci tablicy. Wystarczy więc wywołać ją na rzecz obiektu odzwierciedlającego strukturę dokumentu (zapisanego we właściwości `responseXML`), aby otrzymać dane poszukiwanych węzłów bez konieczności sprawdzania ich typów. Jak to zrobić, pokaże kolejne ćwiczenie.

ĆWICZENIE

4.3. Użycie metody getElementsByTagName

Zmodyfikuj kod ćwiczenia 4.2, tak aby do przetwarzania danych została użyta metoda `getElementsByTagName`.

Treść dokumentu XML pozostanie taka sama jak w poprzednim ćwiczeniu, natomiast kod skryptu będzie miał następującą postać:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<title>Ajax</title>
<script type="text/javascript">
var XMLHttpRequestObject = false;

if(window.XMLHttpRequest){
  XMLHttpRequestObject = new XMLHttpRequest();
}
else if(window.ActiveXObject){
  XMLHttpRequestObject = new ActiveXObject("Microsoft.XMLHTTP");
}

function przetwarzajXML(xml, dest)
{
  var div = document.getElementById(dest);
  var imiona = xml.getElementsByTagName('imie');
  var str = "Imiona odczytane z danych XML: <br />";
  for(var i = 0; i < imiona.length; i++){
    var imie = imiona[i].firstChild.nodeValue;
    str += imie + "<br />";
  }
  div.innerHTML = str;
}
```

```
function pobierzDane(dest)
{
  if(XMLHttpRequestObject){
    var url = "http://localhost/index.xml";
    XMLHttpRequestObject.open("GET", url);

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
          XMLHttpRequestObject.status == 200){
        var xml = XMLHttpRequestObject.responseXML;
        przetwarzajXML(xml, dest);
      }
    }
    XMLHttpRequestObject.send(null);
  }
}
</script>
</head>
<body>
<input type="button" value="Kliknij tu"
       onclick="pobierzDane('dataDiv');" />
<hr>
</div>
<div id="dataDiv">
Ten tekst zostanie zmieniony.
</div>
</body>
</html>
```

Ogólna struktura skryptu jest taka sama jak w ćwiczeniu 4.2, zmianie uległa jednak treść funkcji `przetwarzajXML`. Po pobraniu obiektu warstwy docelowej o identyfikatorze wskazywanym przez argument `dest`:

```
var div = document.getElementById(dest);
```

następuje wywołanie metody `getElementsByTagName` operującej na obiekcie dokumentu XML:

```
var imiona = xml.getElementsByTagName('imie');
```

Argumentem wywołania tej metody jest ciąg znaków określający, jakie elementy (węzły) dokumentu chcemy pobrać (w tym przypadku są to elementy zdefiniowane przez znaczniki `<imie>`), natomiast wynikiem działania jest tablica węzłów. Ta tablica przypisywana jest zmiennej `imiona`. Odczytanie imion odbywa się w pętli `for`, która przebiega wszystkie elementy tablicy:

```
for(var i = 0; i < imiona.length; i++){
```

Ponieważ w każdej komórce znajduje się jeden węzeł zdefiniowany za pomocą znacznika <imie>, to dane dotyczące imienia odnajdziemy w pierwszym węźle potomnym, czyli we właściwości firstChild.nodeValue:

```
var imie = imiona[i].firstChild.nodeValue;
```

Każde odczytane imię jest zapisywane w zmiennej pomocniczej imie, która następnie wykorzystywana jest przy tworzeniu ciągu zapisywanego w zmiennej str:

```
str += imie + "<br />";
```

który po zakończeniu pętli zostanie wyświetlony na warstwie docelowej:

```
div.innerHTML = str;
```