Tehnologii bazate pe XML: parcurgerea documentelor XML folosind PHP

Lect. dr. Teodor Florin Fortiş
Universitatea de Vest Timişoara
Asist. drd. Alexandra Fortiş
Universitatea „Tibiscus" Timişoara

ABSTRACT. XML documents offers an ideal support for transporting data between different applications, even remotely. Parsing and validating XML documents are two of the main activities in using XML documents, together with XML processing by using XSLT. In this paper we offer a point of view on XML parsing by using PHP programming language and two of the basic models used: SAX and DOM.

Tibiscus

1 Introducere

În acest articol este urmărită prezentarea modului în care poate fi realizată integrarea tehnologiilor bazate pe XML în aplicațiile PHP. În timp ce XML oferă posibilități variate de codificare a documentelor în scopul prelucrării ulterioare a structurilor de date conținute în acestea, limbajul PHP oferă suportul independent de platformă pentru transportul datelor depozitate în documentele XML.

Articolul are în vedere doar aspectele de bază ale procesării documentelor XML cu ajutorul limbajului PHP, fără a acoperi XSLT (Extensible Styles Language Transformations), limbaj destinat în principal transformării documentelor XML în alte documente XML.

2 Informații de bază XML

XML (Extensible Markup Language) este un subset al limbajului SGML (Standard Generalized Markup Language). SGML a fost definit în standardul ISO8879 şi a fost realizat pentru a uşura schimbul de documente structurate pe WEB. Spre deosebire de HTML, fişierele XML marchează clar punctul de start şi punctul final al fiecărei părți logice dintr-un document. Fiecare dintre aceste părți logice poartă numele de element.

XML restricționează utilizarea construcțiilor SGML pentru a se asigura împotriva unor situații în care accesul către anumite componente ale documentului nu este posibil. De asemenea, este definit modul în care URL-urile pot fi utilizate pentru a identifica anumite componente ale streamurilor de date XML.

Prin definirea rolului fiecărui element de text într-un model formal, cunoscut sub numele DTD (Document Type Definition), utilizatorii XML pot verifica faptul că fiecare componentă a unui document apare într-o poziție validă în cadrul unui stream de date utilizat pentru schimbul de informații. XML DTD permite de asemenea calculatorului să verifice, de exemplu, ca utilizatorii să nu utilizeze accidental anumite construcții interzise. Astfel de verificări nu sunt posibile atunci când este utilizat un limbaj gen HTML, limbajul clasic utilizat pentru a codifica documentele accesibile pe WEB.

Spre deosebire de SGML, XML nu cere prezența unui DTD. În lipsa acestuia, sistemul XML va utiliza o definiție implicită pentru fiecare dintre componentele nedeclarate din documentul curent.

XML nu a fost creat pentru a standardiza modalitatea de codificare a textului, o asemenea operație ar fi de fapt imposibilă, atâta vreme cât ar fi de aşteptat crearea unei unice scheme de codificare care se va potrivi cu toate limbajele şi aplicațiile. XML poate fi privit mai degrabă ca un limbaj formal care poate fi utilizat pentru a transmite informațiile despre părțile componente ale unui document către un alt sistem de calcul. XML oferă de fapt destulă flexibilitate pentru a putea descrie orice structură logică a unui text.

3 Utilizarea extensiei XML a limbajului PHP

Odată stabilite informațiile de bază legate de extensia XML a limbajul PHP, vom urmări modul în care pot fi utilizate aceste instrumente pentru procesarea şi prezentarea documentelor XML. XSL (Extensible Style

fi.discus

Language) este un instrument puternic care este utilizat de regulă pentru a transforma datele documentelor XML în diferite formate. Totuşi, în continuare vom avea în vedere doar utilizarea SAX (Simple API for XML) şi DOM (Document Object Model), în această din urmă situație fiind oferită o reprezentare arborescentă a structurilor de date XML din document, şi funcțiile de navigare prin acest arbore.

3.1. SAX (Simple API for XML)

Un parser SAX presupune traversarea documentului XML şi apelarea unor funcții specifice pe măsură ce sunt întâlnite diferitele tipuri de etichete, putând fi realizată o distincție clară între etichetele de început, cele de sfârşit şi datele care se găsesc între aceste etichete.

Din punctul de vedere al parserului, lucrurile se rezolvă într-un mod extrem de simplu: sarcina acestuia este numai de a parcurge documentul. Procesarea etichetelor întâlnite cade în sarcina funcțiilor apelate de parser.

3.1.1. Crearea parserului

Primul pas avut în vedere presupune crearea unui parser şi stabilirea tuturor handlerelor necesare pentru procesarea datelor documentului. Secvența tipică avută în vedere este următoarea:

```
<?php
...
$xml_parser = xml_parser_create ();
xml_set_element_handler ($xml_parser,  "processStartElement",
"processEndElement");
xml_set_character_data_handler($xml_parser,
"processCharacterData");
...
?>
```

Functiile processStartElement() şi processEndElement() urmează să fie definite în aplicație, fiind apelate de fiecare dată când sunt întâlnite etichete de deschidere sau etichete de închidere. Datele aflate între aceste etichete urmează să fie procesate prin intermediul funcției processCharacterData(), precizată la rândul ei ca handler.

3.1.2. Procesarea documentului

Următorul pas avut în vedere presupune deschiderea documentului XML,

Tibiscus

citirea şi parcurgerea acestuia cu ajutorul funcţiei xml_parse(). Parserul va apela, de fiecare dată când este întâlnită o etichetă de început sau o etichetă de sfârşit, handlerul corespunzător. Secvenţa tipică de program utilizabilă pentru această parte este următoarea:

```
<?php
...
$xmlFile = "myFile.xml" ;
if (!($xfp = fopen ($xmlFile, "r"))) {   die ("Cannot parse
XML file: $xmlFile"); }

while ($data=fread($xfp, 4096)) {
  if(!xml_parse($xml_parser, $data, feof($xfp))) {
    die (sprintf("XML parsing error: %s (at line %d)",
         xml_error_string(xml_get_error_code($xml_parser),
         xml_get_current_line_number($xml_parser)));
  }
}
xml_parser_free($xml_parser);
?>
```

3.1.3. Un exemplu simplu

Pentru început, descriem un fişier XML simplu pentru prezentarea unei liste de cursuri:

```
<?xml version="1.0"?>
<descrieri>
 <curs>
  <cod>PU0204</cod>
  <titlu>Programare UNIX</titlu>
  <autor>F. Fortiş</autor>
  <durata>14</durata>
  <an_studii>Informatică 4</an_studii>
 </curs>
 <curs>
  <cod>SO0103</cod>
  <titlu>Sisteme de operare</titlu>
  <autor>F. Fortiş</autor>
  <durata>28</durata>
  <an_studii>Informatică 3</an_studii>
 </curs>
 <curs>
  <cod>PI0104</cod>
  <titlu>Prelucrarea imaginilor</titlu>
  <autor>L. Cucu</autor>
```


Fibiscus

```
  <durata>14</durata>
  <an_studii>Informatică 4</an_studii>
 </curs>
</descrieri>
```

Vom prelucra datele din acest exemplu simplu pentru a obține lista cursurilor descrise în format HTML. Întreaga acțiune este rezolvată prin intermediul funcțiilor desemnate ca handlere, asupra cărora ne vom concentra atenția în cele ce urmează.

Acțiunile care urmează să fie avute în vedere pentru fiecare etichetă stabilită sunt, pentru acest exemplu simplu, doar transformări în etichete HTML corect construite. Pentru aceasta construim următoarele tablouri:

```
$elementeStart=array(
  "DESCRIERI"=>"<h2>Lista cursurilor</h2>
    <Table>
      <Tr><Td>Cod curs</Td>
      <Td>Denumire curs</Td>
      <Td>Autor</Td>
      <Td>Durata</Td></Tr>
      <Td>Cui se adreseaza</Td></Tr>",
  "CURS"=>"      <Tr>",
  "COD"=>"      <Td>",
  "TITLU"=>"      <Td>",
  "AUTOR"=>"      <Td>",
  "AN_STUDII"=>"      <Td>") ;
$elementeStop=array("DESCRIERI"=>"</Table>",
  "CURS"=>"</Tr>",
  "COD"=>"</Td>",
  "TITLU"=>"</Td>",
  "AUTOR"=>"</Td>",
  "AN_STUDII"=>"</Td>") ;
```

Cele trei funcții utilizate ca handlere vor urmări, pur şi simplu, rezolvarea mapărilor elementelor fişierului XML pe elemente HTML:

```
function processStartElement ($parser, $nume, $atribute) {
  global $elementeStart;
  if($elementeStart[$nume]) {
    echo $elementeStart[$nume];
  }
}
function processStopElement($parser, $nume) {
  global $elementeStop ;
  if($elementeStop[$nume]) {
    echo $elementeStop[$nume];
```

Tibiscus

```
  }
}
function processCharacterData ($parser, $data) {
  echo $data ;
}
```

Pentru acest exemplu simplu, întâlnirea unei etichete de început/sfârşit este însoțită de o căutare în tabloul "hash" corespunzător. De remarcat însă că nu sunt rezolvate în nici un fel eventualele atribute (inexistente, în acest caz, de altfel) ale elementelor.

Se observă modul extrem de simplu în care sunt tratate documentele XML: parserul parcurge întregul document, apelând funcții specifice fiecărui tip de informații întâlnite: începutul unei etichete, sfârşitul unei etichete sau bucăți de date aflate între etichete. Odată realizată procesarea unei bucăți de informație, prin intermediul parserului se poate trece la următoarea bucată de informație. Totuşi, trebuie remarcat un element extrem de important în tratarea etichetelor: în acest exemplu simplu are loc o transformare a numelor etichetelor astfel încât acestea să fie formate numai din majuscule.

3.2. DOM

O a doua posibilitate de parcurgere a documentelor XML foloseşte DOM, pe baza căruia poate fi realizată construcția unei reprezentări arborescente a structurilor de date din documentele XML, fiind oferite, de asemenea, metode pentru a rezolva navigarea prin acest arbore. Odată găsit un nod particular, pot fi utilizate proprietăți predefinite pentru a obține valoarea unui nod, valoare utilizabilă în scriptul PHP construit.

3.2.1. Crearea arborelui

Pentru a facilita construcția arborelui corespunzător unui document XML, PHP oferă trei funcții distincte: xmldoc (), xmltree () şi xmldocfile () . În timp ce xmldoc () şi xmltree () se aşteaptă să primească un document XML prin intermediul unui şir de caractere, xmldocfile () acceptă un nume de fişier ca argument, construcția arborelui DOM realizându-se pe baza informației depozitate în acest fişier. Modul de funcționare a celor trei tipuri de funcții este pus în evidență în funcțiile următoare:

```
<?php
    $XMLData = "<?xml version=\"1.0\"?" . ">"
```


LibisCS

```
      <test><autor>Myself</autor>
      <varsta>36</varsta></test>";
    $XMLDoc = xmldoc ($XMLData) ;
    $XMLTree = xmltree ($XMLData) ;
?>
```

Observați însă că funcția xmldocfile () este puțin mai pretențioasă în ceea ce privește fișierul sursă: va avea nevoie de calea completă către acesta, atunci când sistemul de operare gazdă este Windows. Pentru a evita o astfel de situație, numele de fișier, chiar şi local, este transformat într-un nume de cale complet folosind funcția realpath ():

```
<?php
    $XMLDataFile = realpath("self.xml");
    $XMLDoc = xmldocfile ($XMLDataFile) ;
?>
```

3.2.2. Informații şi atribute

Obiectul DOM construit prin una dintre cele trei funcții este înzestrat cu metodele şi proprietățile necesare parcurgerii arborelui obținut şi obținerii informațiilor depozitate în noduri. Iată câteva dintre proprietățiile care pot fi utilizate în acest caz:

```
<?php
    echo $XMLTree->version ;
    echo $XMLTree->encoding ;
    echo $XMLTree->standalone ;
    echo $XMLTree->url ;
    echo $XMLTree->charset ;
?>
```

Metodele de bază pentru parcurgerea acestui arbore cuprind următoarele:

```
<?php
    $root = $XMLTree->root() ;
    echo $root->name ;
    $childs = $root->children ();
    /* $childs este un tablou continand toti copiii unui nod
*/
    $firstChild = $childs[0] ;
    echo $firstChild->name ;
    echo $firstChild->content ;
```

Tibiscus

```
    echo $firstChild->type ;
    $parent = $firstChild->parent() ;
?>
```

Toate aceste metode sunt suficiente pentru a putea construi o funcție capabilă să parcurgă documentul XML pentru a construi rezultatul aşteptat.

Pe lângă metodele oferite, în suportul PHP pentru DOM XML sunt incluse şi funcții cu acelaşi efect: domxml_root ($XMLdom) întoarce o referință către nodul rădăcină iar domxml_attributes ($root) este echivalent cu $root->attributes ().

Aceste din urmă funcții sunt extrem de utile în momentul în care elementele documentului XML pot fi însoțite de atribute:

```
<?php
    $XMLData = "<?xml version=\"1.0\"?" . ">"
    <test sex=\"masculin\"><autor>Myself</autor>
      <varsta>36</varsta></test>";

    $XMLDoc = xmldoc ($XMLData) ;
    $radacina = $XMLDoc->root() ;
    $atribute = $radacina->attributes();
    /* Atributul radacinii: perechea sex="masculin"*/
    echo $attributes[0]->name ;
    echo $attributes[0]->content ;
?>
```

3.2.3. Un exemplu simplu

Vom relua acum exemplul avut în vedere în secțiunea pentru a pune în evidență modul în care poate fi utilizat DOM XML pentru a parcurge un document XML.

```
<?php
  $file = "exemplu.xml";
  $eticheteStart=array(                "DESCRIERI"=>"<h2>Lista
cursurilor</h2>
      <Table>
      <Tr><Td>Cod curs</Td>
      <Td>Denumire curs</Td>
      <Td>Autor</Td>
      <Td>Durata</Td></Tr>
      <Td>Cui se adreseaza</Td></Tr>",
     "CURS"=>"      <Tr>",
     "COD"=>"      <Td>",
     "TITLU"=>"      <Td>",
```


Tibiscus

```
    "AUTOR"=>"      <Td>",
    "AN_STUDII"=>"      <Td>") ;
$eticheteStop=array("DESCRIERI"=>"</Table>",
    "CURS"=>"</Tr>",
    "COD"=>"</Td>",
    "TITLU"=>"</Td>",
   "AUTOR"=>"</Td>",
   "AN_STUDII"=>"</Td>") ;
$XMLDom = xmldocfile ($file) ;
$root = $XMLDom->root() ;
$childs = $root->children () ;
printData ($childs);
function printData ($nodes) {
  global $eticheteStart, $eticheteStop;
  for ($e=0; $e<count($nodes);$e++) {
    echo $eticheteStart[$nodes[$e]->name] .
        $nodes[$e]->content ;
    $nodeChilds = getChildren ($nodes[$e]) ;
    printData ($nodeChilds) ;
    echo $eticheteStop[$nodes[$e]->name] ;
  }
}
function getChildren ($node) {
  $tempChilds = $node->children () ;
  $childCollect = array () ;
  $count = 0 ;
  for ($x = 0; $x < count ($tempChilds); $x++) {
    if ($tempChilds[$x]->type == XML_ELEMENT_NODE) {
      $childCollect[$count] = $tempChilds[$x]; $count++ ;
    }
  }
  return $childCollect ;
}
?>
```

3.3. DOM sau SAX?

Folosirea oricăruia dintre cele două modele ne poate conduce către rezultatul aşteptat. Totuşi, există anumite neajunsuri pentru fiecare dintre aceste metode:

- DOM presupune parcurgerea întregului document pentru construirea arborelui. Prin urmare, viteza nu este principalul avantaj pentru DOM față de SAX. SAX, în schimb, presupune o singură parcurgere a documentului.
- SAX oferă o abordare secvențială a problemei: etichetele şi datele sunt procesate îndată de parserul le întâlneşte. Spre deosebire de aceasta,

Tibiscus

DOM oferă posibilitatea de a accesa orice nod al structurii, datorită faptului că arborele este construit înaintea procesării etichetelor.

Bibliografie

[WWWa] XML-RPC.com, http://www.xmlrpc.com

[WWWb] Manual PHP, http://www.php.net/manual

[WWWc] xmlrpc API,http://xmlrpc-epi.sourceforge.net/main.php?t=php_api

[WWWd] W3C DOM, http://www.w3.org/DOM

[WWWe] PHP DOM XML, http://www.php.net/manual/en/ref.xml.php

[WWWf] PHP XML, http://www.phpxml.org

[WWWg] PHP DOM, http://devil.medialab.at

Tibiscus