

PHP, programare obiectuală și XML

Prezentul articol va ilustra implementarea principalelor concepte ale programării obiectuale, în contextul procesării documentelor XML

– Sabin Corneliu Buraga

Programarea orientată-obiect în PHP

Limbajul PHP, mai ales de la versiunea 4, oferă implementarea unora dintre cele mai importante aspecte ale programării obiectuale: încapsularea datelor, moștenirea, polimorfismul. Astfel, PHP dă posibilitatea programatorului să exprime distincția dintre proprietățile generale și cele specifice obiectelor cu care lucrează. Vom ilustra aceste caracteristici prin câteva exemple concludente.

Încapsularea

Încapsularea datelor reprezintă un mecanism de protecție a membrilor de tip dată, accesul la ei realizându-se exclusiv prin intermediul unor metode specifice și nu în mod direct. Acest lucru se realizează prin intermediul claselor, după cum se poate remarca din fragmentul de cod PHP de mai jos:

```
<?php
class Student {
    // date-membru
    var $year;    // an
    var $age;     // vârstă
    var $name;    // nume
    // metode
    function setYear($y) {
        $this->year = $y;
    }
    function getYear() {
        return $this->year;
    }
    ...
}
?>
```

Datele-membru se definesc utilizând `var`, putând avea orice tip (întreg, tablou sau chiar obiect). Metodele se specifică prin declarații de funcții care prelucrează membrii clasei. Pentru a accesa datele clasei, metodele vor trebui să se folosească de construcția `$this->variabila`, altfel variabila se consideră a fi locală în cadrul acelei metode.

Vom crea un obiect prin operatorul `new`, exact ca în limbajul C++:

```
$stud = new Student;
```

Accesarea metodelor clasei se realizează astfel:

```
$stud->setYear(4);
$student_year = $stud->getYear();
```

Din păcate, membrii dată ai clasei pot fi accesați direct, neputându-i declara privați. La fel, metodele nu pot fi specificate private sau protejate, așa cum se întâmplă la C++. Astfel, PHP oferă doar suport pentru încapsulare, dar nu o poate impune.

Pentru funcțiile membru, în loc de construcția sintactică „->”, în PHP 4 se permite „:”. Astfel, codul de mai sus este echivalent cu:

```
$student_year = $stud::getYear();
```

Moștenirea

Moștenirea reprezintă posibilitatea folosirii datelor sau metodelor definite în prealabil de o anumită clasă în cadrul unei clase *derivate* din prima. Relația de derivare se specifică prin cuvântul-cheie `extends`:

```
<?php
class GoodStudent extends Student {
    // date-membru
    var $prizes; // premii
    // metode
    function setPrizes($p) {
        $this->prizes = $p;
    }
    function getPrizes() {
        return $this->prizes;
    }
}
?>
```

Putem scrie următoarele linii:

```
$goodstud = new GoodStudent;
// apel de metodă din clasa de bază
$goodstud->setAge(21);
// apel de metodă din clasa derivată
$goodstud->setPrizes(2);
```

În PHP moștenirea multiplă nu este încă posibilă. Nu putem avea, de exemplu:

```
class GreatStudent extends GoodStudent Genius {
    ...
}
```

Putem însă *redefini* o metodă în clasa derivată. În PHP 3, dacă declarăm un membru-dată în clasa derivată având același nume ca un membru-dată al clasei de bază, atunci nu mai putem accesa membrul-dată din clasa de bază, el fiind „ascuns” de membrul-dată al clasei derivate. În PHP 4, pentru a accesa membri ai clasei părinte ne vom folosi de construcția „:” (exact ca și în C++).

Constructorii

PHP permite specificarea constructorilor, ei fiind metode cu același nume al clasei din care aparțin, fiind apelați automat atunci când instanțiem un obiect al clasei respective. Putem avea de exemplu:

```
<?php
class Student {
    // date-membru
    var $year;      // an
    var $age;       // vârsta
    var $name;     // nume
    // constructor
    function Student($y, $a, $n) {
        $this->year = $y;
        $this->age  = $a;
        $this->name = $n;
    }
    // metode
    function setYear($y) {
        $this->year = $y;
    }
    function getYear() {
        return $this->year;
    }
    ...
}
?>
```

Așadar, acum se poate crea un obiect de genul:

```
$stud = new Student(3, 24, "Radu Filip");
```

Constructorii și metodele, fiind funcții PHP obișnuite, pot avea specificate valori implicite pentru argumente (ca în C++):

```
function Student($y = "4", $a = "22", $n = "")
```

Dacă scriem în acest mod constructorul, atunci în următoarele cazuri vom avea:

```
// year = 4, age = 22, name = ""
$stud = new Student();
// year = 2, age = 22, name = ""
$stud = new Student(2);
// year = 2, age = 20, name = ""
```

```
$stud = new Student(2, 20);
```

Atunci când un obiect al unei clase derivate este creat, numai constructorul lui propriu va fi apelat, constructorul clasei de bază nefiind apelat implicit. Dacă dorim ca și constructorul clasei părinte să fie apelat, o vom face într-o manieră explicită:

```
<?php
function GoodStudent {
    $this->prizes = 3;
    // apel explicit al constructorului clasei de bază
    $this->Student();
}
?>
```

Dacă în PHP 3, constructorii puteau avea orice tip de parametri, începând cu versiunea 4, tipurile permise pentru parametrii unui constructor sunt doar cele simple (întregi, șiruri de caractere), deci nu vor putea fi executate transmiteri de tablouri sau de obiecte.

Dacă o clasă derivată nu posedă propriul ei constructor, va fi apelat implicit constructorul clasei părinte.

Mecanismul obiectual în PHP nu permite declararea destructorilor și nici specificarea de clase abstracte (deși se pot imagina metode mai mult sau mai puțin sofisticate pentru a le simula).

Supraîncărcarea

Supraîncărcarea (asocierea de semantici diferite unei aceleiași funcții pe baza tipurilor parametrilor specificați) nu este suportată nici ea. Putem însă supraîncărca, indirect, constructorii prin crearea de obiecte diferite în funcție de numărul de argumente specificate:

```
<?php
class Student {
    ...
    function Student() {
        // construim un șir de apel
        $method_name = "Student" . func_num_args();
        $this->$method_name();
    }
    function Student1($x) {
```

```
// cod
}
function Student2($x, $y) {
    // cod
}
...
}
?>
```

Vom putea scrie:

```
$stud1 = new Student('1'); // va apela Student1
$stud2 = new Student('1', '2'); // va apela Student2
```

Pentru a pasa un număr variabil de parametri unei funcții și a-i folosi ulterior putem să ne slujim de funcțiile predefinite `func_get_args()`, `func_num_args()` și `func_get_arg()`. Astfel, funcția `Student()` de mai sus va putea afișa toți parametrii transmiși prin codul următor:

```
function Student()
{
    $args_array = func_get_args();
    for ($i = 0; $i < count($args_array); $i++)
        print ($i => $argument_array [$i]);
}
```

Polimorfismul

Polimorfismul reprezintă abilitatea unui obiect de a determina care metodă trebuie invocată pentru un obiect pasat ca argument în momentul rulării și acest lucru se realizează foarte ușor în limbaje interpretate ca PHP.

Vom ilustra acest concept și implementarea lui în PHP presupunând că avem o clasă `Figure` desemnând o figură geometrică în care se definește metoda `draw()` și clasele derivate `Circle` și `Square` unde vom rescrie metoda `draw()` în funcție de figura dorită a fi desenată:

```
<?php
...
function drawing($obj) { // metoda clasei Board
```

```
$obj->draw();
}
// coordonate centru și raza
$circle = new Circle(100, 100, 33);
// coordonate stânga-sus și latura
$square = new Square(100, 200, 74);
// afișează cele două figuri
$board = drawing($circle); // apelează draw() din Circle
$board = drawing($square); // apelează draw() din Square
...
?>
```

Serializarea

PHP nu suportă obiecte serializate (care își păstrează starea și funcționalitatea de-a lungul mai multor invocări ale aplicației, prin intermediul stocării într-un fișier/bază de date și încărcarea lor ulterioară). În PHP prin serializare (funcțiile `serialize()` și `unserialize()`) se vor salva numai membrii-dată, nu și metodele, însă putem serializa tablouri asociative sau indexate, ceea ce reprezintă totuși un avantaj.

Funcții utile pentru manipularea obiectelor

Începând cu PHP 4, se pune la dispoziția programatorului o serie de funcții folositoare:

- `get_class()` va returna numele unui obiect, instanță a unei clase;
- `get_parent_class()` furnizează clasa părinte din care provine un anumit obiect;
- `method_exists()` testează dacă există o metodă pentru un anumit obiect specificat;
- `class_exists()` testează existența unei clase;
- `is_subclass_of()` va determina existența unei relații de moștenire dintre două clase.

O altă facilitare este cea a transmiterii prin referință a parametrilor și nu prin valoare, cum se realizează în mod implicit. Pentru a fi transmis prin referință, vom prefixa numele aceluiași parametru cu caracterul ampersand „&”.

PHP și XML

Cele de mai sus ne permit să realizăm o procesare elegantă a documentelor XML, folosind SAX ori DOM.

Utilizarea lui Expat

Elaborat de *James Clark*, procesorul Expat este deja încorporat în serverul Apache începând cu versiunea 1.3.9, iar în PHP este inclus de la versiunea 3.0.6. Analiza XML este bazată pe evenimente, fiecare tip de nod al arborelui asociat documentului XML declanșând un anumit eveniment care va trebui tratat de o funcție definită de programator. Pentru a atașa funcții evenimentelor XML, ne vom folosi de o serie de funcții predefinite:

- `xml_set_element_handler()` stabilește funcțiile care vor fi apelate pentru procesarea elementelor XML (pentru tag-urile de început și de sfârșit);
- `xml_set_character_data_handler()` stabilește funcția care va fi apelată atunci când analizorul întâlnește un nod de tip CDATA (text);
- `xml_set_processing_instruction_handler()` definește funcția care va fi executată la apariția unei instrucțiuni de procesare.

Alte funcții importante puse la dispoziție sunt:

- `xml_parser_create()` inițializează analizorul XML și returnează o instanță a sa;
- `xml_parser_free()` eliberează memoria alocată analizorului;
- `xml_set_object()` stabilește adresele funcțiilor care vor fi utilizate de analizor pentru a realiza procesarea documentului XML dorit;
- `xml_parser_set_option()` se folosește la setarea unor opțiuni de analiză XML (e.g. modul de tratare a scrierii cu majuscule sau minuscule a tag-urilor);
- `xml_get_error_code()` furnizează codul de eroare în urma eșecului procesării.

Pot fi amintite, de asemeni, funcțiile dând mai multe amănunte despre erorile survenite în timpul analizei: `xml_error_string()` și `xml_get_current_line_number()`. Funcția `xml_parse()` returnează, în caz de eșec, o serie de coduri de eroare ale căror constante simbolice predefinite pot fi consultate în manualul PHP.

Vom defini o clasă pe care o vom folosi ulterior la procesarea documentelor XML (vom salva acest cod în fișierul `parseXML.php`).

```
<?php
// o clasă pentru prelucrarea documentelor XML
class parseXML {
    var $xml_parser; /* instanța analizorului XML */
    var $xml_file; /* numele fișierului XML */
    var $html_code; /* codul HTML generat */
    var $open_tags; /* mulțimea tag-urilor de început */
    var $close_tags; /* mulțimea tag-urilor de sfârșit */
// constructor
function parseXML() {
    $this->xml_parser = "";
    $this->xml_file = "";
    $this->html_code = "";
    $this->open_tags = array();
    $this->close_tags = array();
}
// destructor
function destroy() {
    if ($this->xml_parser)
        xml_parser_free($this->xml_parser);
}
// metode
// setează tag-urile de început
function set_open_tags($tags) {
    $this->open_tags = $tags;
}
// setează tag-urile de sfârșit
function set_close_tags($tags) {
    $this->close_tags = $tags;
}
}
```

```
// setează numele fișierului XML
function set_xml_file($file) {
    $this->xml_file = $file;
}
// furnizează codul HTML generat
function get_html_code() {
    return $this->html_code;
}
// tratarea evenimentului de
// apariție a unui tag de început
function start_element($parser, $name, $attrs) {
    if ($format = $this->open_tags[$name])
        $this->html_code .= $format;
}
// tratarea evenimentului de
// apariție a unui tag de sfârșit
function end_element($parser, $name) {
    if ($format = $this->close_tags[$name])
        $this->html_code .= $format;
}
// tratarea evenimentului de
// apariție a unui element de tip CDATA
function character_data($parser, $data) {
    $this->html_code .= $data;
}
// tratarea evenimentului de
// apariție a unei instrucțiuni de procesare
function processing_instruction($parser, $target, $data) {
    switch (strtolower($target)) {
        case "php": eval($data);
        break;
    }
}
// funcția de analiză propriu-zisă
function parse() {
    // instanțiază procesorul XML
    $this->xml_parser = xml_parser_create();
    // înregistrează funcțiile de analiză
    xml_set_object($this->xml_parser, &$this);
    // seteaza opțiunile de analiză
    // (tag-urile nu sunt rescrise cu caractere mari)
    xml_parser_set_option($this->xml_parser,
XML_OPTION_CASE_FOLDING, false);
    // setează funcțiile de procesare a elementelor XML
    xml_set_element_handler($this->xml_parser,
        "start_element", "end_element");
    xml_set_character_data_handler($this->xml_parser,
        "character_data");
    xml_set_processing_instruction_handler($this->xml_parser,
        "processing_instruction");
    // deschide fișierul XML
    if (!$fp = fopen($this->xml_file, "r"))
        die("could not open XML source");
    // procesează fișierul
    while ($data = fread($fp, 4096)) {
        if (!xml_parse($this->xml_parser, $data, feof($fp))) {
            // eroare de procesare
            die(sprintf("XML error: %s at line %d",
                xml_error_string(xml_get_error_code(
                    $this->xml_parser)),
                xml_get_current_line_number($this->xml_parser)));
        }
    } /* while */
} /* parse() */
```

```

} /* class */
?>

```

Folosind această clasă, putem transforma un document XML în cod HTML, după cum se poate remarca din exemplul de mai jos, unde va fi prelucrat un fișier XML conținând impresii despre un anumit site Web:

```

<?php
// necesită prezența clasei definite mai sus
require("parseXML.php");
// substituția tag-urilor XML cu cod HTML
// se folosesc două tablouri asociative
$open_tags = array(
    "impresii" => "\n<!-- generat de parseXML -->\n" .
        "<table cellpadding=\\"5\" align=\\"center\"
        border=\\"1\">",
    "impresie" => "<tr align=\\"center\">",
    "nume" => "<td><h4>",
    "ocupatia" => "<td><p style=\\"color: blue\">",
    "virsta" => "<td><p><i>",
    "text" => "<td bgcolor=\\"#EEEEEE\"><p
        align=\\"justify\">");
$close_tags = array(
    "impresii" => "</table>\n" .
        "<!-- sfârșitul generării parseXML -->\n",
    "impresie" => "</tr>",
    "nume" => "</h4></td>",
    "ocupatia" => "</p></td>",
    "virsta" => "</i></p></td>",
    "text" => "</p></td>");
// instanțiază și inițializează analizorul
$parser = new parseXML();
$parser->set_xml_file("impresii.xml");
$parser->set_open_tags($open_tags);
$parser->set_close_tags($close_tags);
// rulează analizorul
$parser->parse();
// afișează rezultatul
echo $parser->get_html_code();
// distruge obiectul
$parser->destroy();
?>

```

Clasa definită este suficient de generală pentru a putea fi utilizată pentru orice tip de document XML. Tag-urile netratate de programul nostru vor fi ignorate.

O varianta extinsa e disponibila in cartea
 'Tehnologii Web', Matrix Rom, Bucuresti, 2001:
 www.infoiasi.ro/~busaco/books/web.html
 Vezi si http://www.infoiasi.ro/~phpapps/

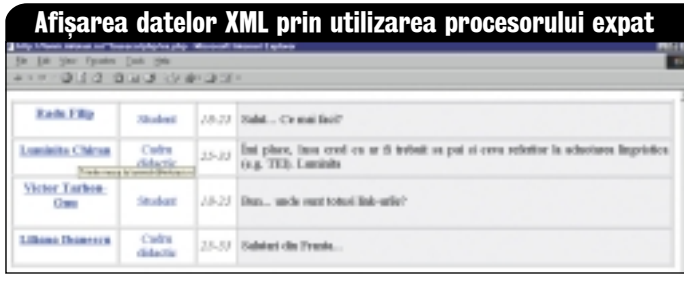
De multe ori însă ar fi de dorit să realizăm anumite prelucrări asupra datelor stocate de fișierele XML. Putem încă să ne slujim de parseXML. De exemplu, am dori ca utilizatorul (sau autorul site-ului) să poată trimite mesaje celor care și-au lăsat impresiile. Pentru aceasta vom defini o clasă derivată din clasa parseXML și vom redefini funcțiile start_element() și end_element():

```

<?php
    require("parseXML.php");
// folosirea moștenirii pentru a defini un alt comportament
class parseXML2 extends parseXML {
// indică dacă există atributul "email"
    var $is_email = 0;
// redefinirea metodelor
    function start_element($parser, $name, $attrs) {
        // apelează metoda din clasa de bază
        parseXML::start_element($parser, $name, $attrs);
        // pune și link spre adresa e-mail
        if (!strcmp($name, "nume")) {
            if ($attrs["email"]) {
                $format = "<a title=\\"Trimitte mesaj la " .
                    $attrs["email"] .
                    "\" href=\\"mailto:" . $attrs["email"] . "\">";
                $this->html_code .= $format;
                $this->is_email = 1;
            }
            else
                $this->is_email = 0;
        }
    }
    function end_element($parser, $name) {
        // închide
        if (!strcmp($name, "nume")) {
            if ($this->is_email) {
                $format = "";
                $this->html_code .= $format;
            }
        }
        // apelează metoda din clasa de bază
        parseXML::end_element($parser, $name);
    }
}
?>

```

Noul membru de tip dată is_email este folosit pentru a putea închide corect tag-urile elementului <a> (se poate întâmpla ca atributul email să



nu apară). Restul codului rămâne același, în loc de \$parser = new parseXML() trebuind a fi scrisă linia \$parser = new parseXML2().

O posibilă rulare a scriptului PHP de mai sus poate avea ca efect următoarea pagină Web:

Desigur, dacă facem numai prelucrări asupra documentelor XML, ne putem dispensa de definirea celor două tablouri asociative open_tags[] și close_tags[], iar în funcțiile startElement() și endElement() putem insera orice cod dorim.

Utilizând tehnicile descrise mai sus, studenții *Constantin Gheorghiuță*, *Valentin Păscăreanu* și *Dan Țorin* au realizat o aplicație Web bazată pe PHP pentru managementul lucrărilor de licență la Facultatea de Informatică a Universității „Al.I.Cuza” din Iași. Datele referitoare la licență sunt stocate într-un fișier XML având formatul de mai jos (lăsăm cititorului plăcerea de a construi DTD-ul pentru validarea acestui document):

```
<database>
  <record>
    <!-- Numele și contul studentului -->
    <stud cont="..."> ... </stud>
    <!-- Numele și contul coordonatorului -->
    <prof cont="..."> ... </prof>
    <!-- Tema lucrării de licență -->
    <tema> ... </tema>
    <!-- Descrierea (opțională) a temei -->
    <desc> ... </desc>
    <!-- Legături (URI-uri) relevante -->
    <link> ... </link>
    ...
  </record>
</database>
```

PHP și libxml

PHP 4.0 include analizorul libxml elaborat de *Daniel Veillard* și integrat în motorul PHP de *Uwe Steinman* (vezi și capitolul referitor la DOM). PHP funcționează cu libxml-2.0.0 sau o versiune superioară.

Arborele abstract asociat unui document XML va putea fi creat de una dintre funcțiile:

- xmlrpc() - va genera arborele pornind de la un șir de caractere reprezentând un document XML;
- xmlrpcfile() - va încărca un document XML de pe disc și va construi arborele;
- new_xmlrpc() - va genera un arbore vid.

Arborele DOM va fi reprezentat în PHP printr-un obiect aparținând clasei speciale „DOM document” având proprietățile doc (resursă), version (șir de caractere, „1.0” în prezent) și type (întreg lung). Sunt puse la dispoziție următoarele metode:

- root() - returnează nodul rădăcină al arborelui DOM;
- addroot() - adaugă un nod rădăcină la un arbore vid creat de new_xmlrpc();
- dtd() - returnează un obiect aparținând clasei DTD, care nu posedă metode, ci numai membrii-dată name (numele elementului rădăcină al documentului XML), sysid (conține un identificator sistem al DTD-ului asociat documentului, e.g. impresii.dtd) și extid (reprezintă un identificator extern);

- dumpmem() - convertește în șir de caractere reprezentarea internă a arborelui DOM.

Un script PHP care va genera documentul XML:

```
<?xml version="1.0" ?>
<nume>Sabin-Corneliu Buraga</nume>

// un nou arbore DOM
$doc = new_xmlrpc("1.0");
// inserează nodul-rădăcină
$root = $doc->add_root("nume");
// adaugă nodului un conținut
$root->content = "Sabin-Corneliu Buraga";
// afișează documentul XML generat
print (htmlspecialchars($doc->dumpmem()));
```

va fi următorul:

După cum se poate remarca, putem foarte ușor construi prin program documente sau fragmente de documente XML, ceea ce nu se putea cu expat.

După cum am văzut, în cadrul modelului DOM orice componentă a unui document XML va fi reprezentată prin intermediul unui nod al arborelui asociat. Un obiect de tip nod va avea metodele:

- parent() - desemnează nodul părinte al nodului curent;
- children() - returnează nodurile copii ale nodului curent;
- attributes() - furnizează atributele asociate unui nod de tip element;
- new_child() - generează un nod copil;
- getattr() - returnează valoarea unui atribut, dacă există;

Constantele predefinite desemnând tipurile de noduri DOM

Constantă	Valoare
XML_ELEMENT_NODE	1
XML_ATTRIBUTE_NODE	2
XML_TEXT_NODE	3
XML_CDATA_SECTION_NODE	4
XML_ENTITY_REF_NODE	5
XML_ENTITY_NODE	6
XML_PI_NODE	7
XML_COMMENT_NODE	8
XML_DOCUMENT_NODE	9
XML_DOCUMENT_TYPE_NODE	10
XML_DOCUMENT_FRAG_NODE	11
XML_NOTATION_NODE	12

- setattr() - modifică valoarea unui atribut. Sunt disponibili și următorii membri-dată:
- type - desemnează tipul de nod; pentru o manevrare mai facilă a tipurilor nodurilor sunt predefinite constantele din tabel.

Aceste constante se pot folosi și în programele C folosind biblioteca libxml.

- name - reprezintă numele nodului (e.g. numele unui element XML);
- content - desemnează conținutul unui anumit nod (dacă există).

Pentru a parcurge întreg arborele de noduri sau numai părți din el ne putem folosi de xmltree(), funcție care va analiza documentul XML dat ca parametru (sub formă de șir de caractere) și va returna o structură de obiecte PHP reprezentând acel document. Această structură nu va putea fi însă modificată.

Domnul Sabin-Corneliu Buraga este doctorand în Computer Science la Universitatea „Al.I.Cuza” din Iași și poate fi contactat la adresa busaco@infoiasi.ro. ■ 66