This second article in a three-part series will discuss XML parsing techniques of PHP5, focusing on parsing large or complex XML documents. It will offer some background about parsing extensions and, specifically, what parsing methods are best suited to what types of XML documents and why.

View more content in this series

Introduction

PHP5 offers an improved variety of XML parsing techniques. James Clark's Expat SAX parser, now based on libxml2, is no longer the only fully functional game in town. Parsing with the DOM, fully compliant with the W3C standard, is a familiar option. Both SimpleXML, which you saw in Part 1 (see Related topics), and XMLReader, which is easier and faster than SAX, offer additional parsing approaches. All the XML extensions are now based on the libxml2 library by the GNOME project. This unified library allows for interoperability between the different extensions. This article will cover PHP5's XML parsing techniques, focusing on parsing large or complex XML documents. It will offer some background about parsing techniques, what method is best suited to what types of XML documents, and, if you have a choice, what your criteria for choosing should be.

SimpleXML

Other articles in this series

- XML for PHP developers, Part 3: Advanced techniques to read, manipulate, and write XML

Part 1 provided essential information on XML and focused on quick-start Application Programming Interfaces (APIs). It demonstrated how SimpleXML, combined with the Document Object Model (DOM) as necessary, is the ideal choice if you work with straightforward, predictable, and relatively basic XML documents.
XML and PHP5

Extensible Markup Language (XML) is described as both a markup language and a text-based data storage format; it offers a text-based means to apply and describe a tree-based structure to information.

In PHP5, there are totally new and rewritten extensions for parsing XML. Those that load the entire XML document into memory include SimpleXML, the DOM, and the XSLT processor. Those parsers that provide you with one piece of the XML document at a time include the Simple API for XML (SAX) and XMLReader. SAX functions the same way it did in PHP4, but it's not based on the expat library anymore, but on the libxml2 library. If you are familiar with the DOM from other languages, you will have an easier time coding with the DOM in PHP5 than previous versions.

XML parsing fundamentals

The two basic means to parse XML are: tree and stream. Tree style parsing involves loading the entire XML document into memory. The tree file structure allows for random access to the document's elements and for editing of the XML. Examples of tree-type parsing include the DOM and SimpleXML. These share the tree-like structure in different but interoperable formats in memory. Unlike tree style parsing, stream parsing does not load the entire XML document into memory. The use of the term stream in this context corresponds closely to the term stream in streaming audio. What it is doing and why it is doing it is exactly the same, namely delivering a small amount of data at a time to preserve both bandwidth and memory. In stream parsing, only the node currently being parsed is accessible, and editing the XML, as a document, is not possible. Examples of stream parsers include XMLReader and SAX.

Tree-based parsers

Tree-based parsers are so named because they load the entire XML document into memory with the document root being the trunk, and all children, grandchildren, subsequent generations, and attributes being the branches. The most familiar tree-based parser is the DOM. The easiest tree-based parser to code is SimpleXML. You will look at both.

Parsing with the DOM

The DOM standard, according the W3C, is "... a platform and language neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents." The libxml2 library by the GNOME project implements the DOM, along with all its methods, in C. Since all of the PHP5 XML extensions are based on libxml2, there is complete interoperability between the extensions. This interoperability greatly enhances their functionality. You can, for instance, use XMLReader, a stream parser, to get an element, import it into the DOM and extract data using XPath. That is a lot of flexibility. You'll see this in Listing 5.

The DOM is a tree-based parser. The DOM is easy to understand and utilize since its structure in memory resembles the original XML document. DOM passes on information to the application by creating a tree of objects that duplicates exactly the tree of elements from the XML file, with
every XML element being a node in the tree. DOM is a W3C standard, which gives the DOM a lot of authority with developers due to its consistency with other programming languages. Because the DOM builds a tree of the entire document, it uses a lot of memory and processor time.

The DOM in action

If you're forced by your design or any other constraint to be a one trick pony in the area of parsers, this is where you want to be due to flexibility alone. With the DOM, you can build, modify, query, validate and transform XML Documents. You can use all DOM methods and properties. Most DOM level 2 methods are implemented with properties properly supported. Documents parsed with the DOM can be as complex as they come because of its tremendous flexibility. Remember, however, that flexibility comes at a price if you load a large XML document into memory all at once.

The example in Listing 1 uses the DOM to parse the document and retrieves an element with getElementById. It's necessary to validate the document by setting validateOnParse=true before referring to the ID. According to the DOM standard, this requires a DTD which defines the attribute ID to be of type ID.

Listing 1. Using the DOM with a basic document

```php
<?php
$doc = new DomDocument;
// We must validate the document before referring to the id
$doc->validateOnParse = true;
$doc->Load('basic.xml');
echo "The element whose id is myelement is: " .
$doc->getElementById('myelement')->tagName . "\n";
?>
```

The getElementsByTagName() function returns a new instance of class DOMNodeList containing the elements with a given tag name. The list, of course, has to be walked through. Altering the document structure while iterating the NodeList returned by getElementsByTagName() affects the NodeList you are iterating (see Listing 2). There is no validation requirement.

Listing 2. DOM getElementsByTagName method

```php
DOMDocument {
   DOMNodeList getElementsByTagName(string name);
}
```

The example in Listing 3 uses the DOM with XPath.

Listing 3. Using the DOM and parsing with XPath

```php
<?php
$doc = new DOMDocument;
```
// We don't want to bother with white spaces
$doc->preserveWhiteSpace = false;
$doc->Load('book.xml');

$xpath = new DOMXPath($doc);
// We start from the root element
$query = '//book/chapter/para/informaltable/tgroup/tbody/row/entry[. = "en"]';
$entries = $xpath->query($query);
foreach ($entries as $entry) {
    echo "Found {$entry->previousSibling->previousSibling->nodeValue}, " .
         "by {$entry->previousSibling->nodeValue}\n";
}
?>

Having said all of those nice things about the DOM, I'm going to wind up with an example of what not to do with the DOM just to make the point as strongly as possible, and, then, in the next example, how to save yourself. Listing 4 illustrates loading a large file into the DOM simply to extract the data from a single attribute with DomXpath.

Listing 4. Using the DOM with XPath the wrong way, on a large XML document

<?php
// Parsing a Large Document with DOM and DomXpath
// First create a new DOM document to parse
$dom = new DomDocument();

// This document is huge and we don't really need anything from the tree
// This huge document uses a huge amount of memory
$dom->load("tooBig.xml");
$xp = new DomXPath($dom);
$result = $xp->query("/blog/entries/entry[@ID = 5225]/title") ;
print $result->item(0)->nodeValue ."\n";
?>

This final, follow-up example in Listing 5 uses the DOM with XPath in the same way, except the data is passed one element at a time by XMLReader using expand(). With this method, you can convert a node passed by XMLReader to a DOMElement.

Listing 5. Using the DOM with XPath the right way, on a large XML document

<?php
// Parsing a large document with XMLReader with Expand - DOM/DOMXpath
$reader = new XMLReader();
$reader->open("tooBig.xml");
while ($reader->read()) {
    switch ($reader->nodeType) {
        case (XMLREADER::ELEMENT):
            if ($reader->localName == "entry") {
                if ($reader->getAttribute("ID") == 5225) {
                    $node = $reader->expand();
                    $dom = new DomDocument();
                    $n = $dom->importNode($node,true);
                }
            }
    }
}
Parsing with SimpleXML

The SimpleXML extension is another choice for parsing an XML document. The SimpleXML extension requires PHP5 and includes built-in XPath support. SimpleXML works best with uncomplicated, basic XML data. Provided that the XML document isn't too complicated, too deep, and lacks mixed content, SimpleXML is simpler to use than the DOM, as its name implies. It is more intuitive if you are working with a known document structure.

SimpleXML in action

SimpleXML shares many of the advantages of the DOM and is more easily coded. It allows easy access to an XML tree, has built-in validation and XPath support, and is interoperable with the DOM, giving it read and write support for XML documents. You can code documents parsed with SimpleXML simply and quickly. Remember however, that, like the DOM, SimpleXML comes with a price for its ease and flexibility if you load a large XML document into memory.

The following code in Listing 6 extracts <plot> from the example XML.

Listing 6. Extracting the plot text

```php
$xmlstr = <<<XML
<?xml version='1.0' standalone='yes'?>
<books>
  <book>
    <title>Great American Novel</title>
    <plot>
      Cliff meets Lovely Woman. Loyal Dog sleeps, but wakes up to bark at mailman.
    </plot>
    <success type="bestseller">4</success>
    <success type="bookclubs">9</success>
  </book>
</books>
XML;

$xml = new SimpleXMLElement($xmlstr);
echo $xml->book[0]->plot; // "Cliff meets Lovely Woman. ..."
```

On the other hand, you might want to extract a multi-line address. When multiple instances of an element exist as children of a single parent element, normal iteration techniques apply. The following code in Listing 7 demonstrates this functionality.
Listing 7. Extracting multiple instances of an element

```php
<?php
$xmlstr = <<<XML
<xml version='1.0' standalone='yes'>
<books>
    <book>
        <title>Great American Novel</title>
        <plot>
            Cliff meets Lovely Woman.
        </plot>
        <success type="bestseller">4</success>
        <success type="bookclubs">9</success>
    </book>
    <book>
        <title>Man Bites Dog</title>
        <plot>
            Reporter invents a prize-winning story.
        </plot>
        <success type="bestseller">22</success>
        <success type="bookclubs">3</success>
    </book>
</books>
XML;
?>

<?php
$xml = new SimpleXMLElement($xmlstr);
foreach ($xml->book as $book) {
    echo $book->plot, '<br />
};

In addition to reading element names and their values, SimpleXML can also access element attributes. In the code shown in Listing 8, you access attributes of an element just as you would elements of an array.

Listing 8. Demonstrating SimpleXML accessing the attributes of an element

```php
<?php
$xmlstr = <<<XML
<?xml version='1.0' standalone='yes'>
<?xml version='1.0' standalone='yes'>
<books>
    <book>
        <title>Great American Novel</title>
        <plot>
            Cliff meets Lovely Woman.
        </plot>
        <success type="bestseller">4</success>
        <success type="bookclubs">9</success>
    </book>
    <book>
        <title>Man Bites Dog</title>
        <plot>
            Reporter invents a prize-winning story.
        </plot>
        <success type="bestseller">22</success>
        <success type="bookclubs">3</success>
    </book>
</books>
XML;
?>

<?php
$xml = new SimpleXMLElement($xmlstr);
foreach ($xml->book as $book) {
;
```
<?php
$xml = new SimpleXMLElement($xmlstr);
foreach ($xml->book[0]->success as $success) {
    switch((string) $success['type']) {
    case 'bestseller':
        echo $success, ' months on bestseller list<br />
    break;
    case 'bookclubs':
        echo $success, ' bookclub listings<br />
    break;
    }
}
?>

This final example (see Listing 9) uses SimpleXML and the DOM with XMLReader. With XMLReader, the data is passed one element at a time using expand(). With this method, you can convert a node passed by XMLReader to a DOMElement, and then to SimpleXML.

**Listing 9. Using SimpleXML with the DOM and XMLReader to parse a large XML document**

```php
<?php
// Parsing a large document with Expand and SimpleXML
$reader = new XMLReader();
$reader->open("tooBig.xml");
while (!$reader->read()) {
    switch ($reader->nodeType) {
    case (XMLREADER::ELEMENT):
        if ($reader->localName == "entry") {
            if ($reader->getAttribute("ID") == 5225) {
                $node = $reader->expand();
                $dom = new DomDocument();
                $n = $dom->importNode($node,true);
                $dom->appendChild($n);
                $sxe = simplexml_import_dom($n);
                echo $sxe->title;
            }
        }
    }
}
?>
```

**Stream-based parsers**

Stream-based parsers are so named because they parse the XML in a stream with much the same rationale as streaming audio, working with a particular node, and, when they are finished with that node, entirely forgetting its existence. XMLReader is a pull parser and you code for it in much the same way as for a database query result table in a cursor. This makes it easier to work with unfamiliar or unpredictable XML files.

**Parsing with XMLReader**

The XMLReader extension is a stream-based parser of the type often referred to as a cursor type or pull parser. XMLReader pulls information from the XML document on request. It is based on
XMLReader in action

XMLReader, as a stream parser, is well-suited to parsing large XML documents; it is a lot easier to code than SAX and usually faster. This is your stream parser of choice.

This example in Listing 10 parses a large XML document with XMLReader.

Listing 10. XMLReader with a large XML file

```php
<?php
$reader = new XMLReader();
$reader->open("tooBig.xml");
while ($reader->read()) {
    switch ($reader->nodeType) {
    case (XMLREADER::ELEMENT):
        if ($reader->localName == "entry") {
            if ($reader->getAttribute("ID") == 5225) {
                while ($reader->read()) {
                    if ($reader->nodeType == XMLREADER::ELEMENT) {
                        if ($reader->localName == "title") {
                            $reader->read();
                            echo $reader->value;
                            break;
                        }
                        if ($reader->localName == "entry") {
                            break;
                        }
                    }
                    break;
                }
            }
        }
    }
}
?>
```

Parsing with SAX

The Simple API for XML (SAX) is a stream parser. Events are associated with the XML document being read, so SAX is coded in callbacks. There are events for element opening and closing tags, for the content of elements, for entities, and for parsing errors. The primary reason to use the SAX parser rather than the XMLReader is that the SAX parser is sometimes more efficient and usually more familiar. A major disadvantage is that SAX parser code is complex and more difficult to write than XMLReader code.

SAX in action

SAX is likely familiar to those who worked with XML in PHP4, and the SAX extension in PHP5 is compatible with the version they're used to. Since it's a stream parser, it's a good choice for large files, but not as good a choice as XMLReader.
This example in Listing 11 parses a large XML document with SAX.

**Listing 11. Using SAX to parse a large XML file**

```php
Listing 11. Using SAX to parse a large XML file

<?php

//This class contains all the callback methods that will actually
//handle the XML data.
class SaxClass {
    private $hit = false;
    private $titleHit = false;

    //callback for the start of each element
    function startElement($parser_object, $elementname, $attribute) {
        if ($elementname == "entry") {
            if ( $attribute['ID'] == 5225) {
                $this->hit = true;
            } else {
                $this->hit = false;
            }
        }
        if ($this->hit & $elementname == "title") {
            $this->titleHit = true;
        } else {
            $this->titleHit =false;
        }
    }

    //callback for the end of each element
    function endElement($parser_object, $elementname) {
    }

    //callback for the content within an element
    function contentHandler($parser_object,$data) {
        if ($this->titleHit) {
            echo trim($data)."<br />";
        }
    }

    //Function to start the parsing once all values are set and
    //the file has been opened
    function doParse($parser_object) {
        if (!($fp = fopen("tooBig.xml", "r")));

        //loop through data
        while ($data = fread($fp, 4096)) {
            //parse the fragment
            xml_parse($parser_object, $data, feof($fp));
        }
    }

    $SaxObject = new SaxClass();
    $parser_object = xml_parser_create();
    xml_set_object ($parser_object, $SaxObject);
    //Don't alter the case of the data
    xml_parser_set_option($parser_object, XML_OPTION_CASE_FOLDING, false);
    xml_set_element_handler($parser_object,"startElement","endElement"); xml_set_character_data_handler($parser_object, "contentHandler");
    doParse($parser_object);
?>
```
Summary

PHP5 offers an improved variety of parsing techniques. Parsing with the DOM, now fully compliant with the W3C standard, is a familiar option, and is your choice for complex but relatively small documents. SimpleXML is the way to go for basic and not-too-large XML documents, and XMLReader, easier and faster than SAX, is the stream parser of choice for large documents.
Related topics

- **XML for PHP developers, Part 1: The 15-minute PHP-with-XML starter** (Cliff Morgan, developerWorks, February 2007): In the first article of this three-part series, discover PHP5’s XML implementation and how easy it is to work with XML in a PHP environment.

- **XML for PHP developers, Part 3: Advanced techniques to read, manipulate, and write XML** (Cliff Morgan, developerWorks, March 2007): Learn more techniques to read, manipulate, and write XML in PHP5 in this final article of a three-part series on XML for PHP developers.

- **Reading and writing the XML DOM in PHP** (Jack Herrington, developerWorks, December 2005): Explore three methods to reading XML: the DOM library, the SAX parser, and regular expressions. Also, look at how to write XML using DOM and PHP text templating.

- **What kind of language is XSLT** (Michael Kay, developerWorks, April 2005): Put XSLT in context as you learn where the language comes from, what it's good at, and why you should use it.

- **Tip: Implement XMLReader: An interface for XML converters** (Benoît Marchal, developerWorks, November 2003): In this tip, explore APIs for XML pipelines and find why the familiar XMLReader interface is appropriate for many XML components.

- **SimpleXML Processing with PHP** (Elliotte Rusty Harold, developerWorks, October 2006): Try the SimpleXML extension and enable your PHP pages to query, search, modify, and republish XML.

- **Introducing Simple XML in PHP5** (Alejandro Gervasio, Dev Shed, June 2006): In the first of a three-part article series on SimpleXML, save work with the basics of the simplexml extension in PHP 5, a library that primarily focuses on parsing simple XML files.

- **PHP Cookbook, Second Edition** (Adam Trachtenberg and David Sklar, O'Reilly Media, August 2006): Learn to build dynamic Web applications that work on any Web browser.

- **XML.com**: Visit O'Reilly's XML site for comprehensive coverage of the XML world.

- **W3C XML Information**: Read the XML specification from the source.

- **PHP development home site**: Learn more about this widely-used general-purpose scripting language that is especially suited for Web development.

- **Visit PEAR: PHP Extension and Application Repository**: Get more information on PEAR, a framework and distribution system for reusable PHP components.

- **PECL: PHP Extension Community Library**: Visit the sister site to PEAR and repository for PHP Extensions.

- **Planet PHP**: Visit the PHP developer community news source.

- **xmllib2**: Get the the XML C parser and toolkit of Gnome.

- **IBM certification**: Find out how you can become an IBM-Certified Developer.

- **XML technical library**: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.

- **IBM trial software**: Build your next development project with trial software available for download directly from developerWorks.