Parsing XML With SimpleXML

(http://www.sitepoint.com/author/spanda/)

Parsing XML essentially means navigating through an XML document and returning the relevant data. An increasing number of web services return data in JSON format, but a large number still return XML, so you need to master parsing XML if you really want to consume the full breadth of APIs available.

Using PHP's SimpleXML extension that was introduced back in PHP 5.0, working with XML is very easy to do. In this article I'll show you how.

Basic Usage

Let's start with the following sample as `languages.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<languages>
 <lang name="C">
  <appeared>1972</appeared>
  <creator>Dennis Ritchie</creator>
 </lang>
 <lang name="PHP">
  <appeared>1995</appeared>
  <creator>Rasmus Lerdorf</creator>
 </lang>
 <lang name="Java">
  <appeared>1995</appeared>
  <creator>James Gosling</creator>
 </lang>
</languages>
```

The above XML document encodes a list of programming languages, giving two details about each language: its year of implementation and the name of its creator.

The first step is to loading the XML using either `simplexml_load_file()` or `simplexml_load_string()`. As you might expect, the former will load the XML file a file and the later will load the XML from a given string.

```
1 <?php
2 $languages = simplexml_load_file("languages.xml");
```

Both functions read the entire DOM tree into memory and returns a `SimpleXMLElement` object representation of it. In the above example, the object is stored into the `$languages` variable. You can then use `var_dump()` or `print_r()` to get the details of the returned object if you like.

```
SimpleXMLElement Object
(
    [lang] => Array
        (
            [0] => SimpleXMLElement Object
                (
                    [@attributes] => Array
                        (
                            [name] => C
                        )
                    [appeared] => 1972
                    [creator] => Dennis Ritchie
                )
            [1] => SimpleXMLElement Object
                (
                    [@attributes] => Array
                        (
                            [name] => PHP
                        )
                    [appeared] => 1995
                    [creator] => Rasmus Lerdorf
                )
            [2] => SimpleXMLElement Object
                (
                    [@attributes] => Array
                        (
                            [name] => Java
                        )
                    [appeared] => 1995
                    [creator] => James Gosling
                )
        )
)
```

The XML contained a root `language` element which wrapped three `lang` elements, which is why the `SimpleXMLElement` has the public property `lang` which is an array of three `SimpleXMLElements`. Each element of the array corresponds to a `lang` element in the XML document.

You can access the properties of the object in the usual way with the `->` operator. For example, `$languages->lang[0]` will give you a `SimpleXMLElement` object which corresponds to the first `lang` element. This object then has two public properties: `appeared` and `creator`.

```
<?php
$languages->lang[0]->appeared;
$languages->lang[0]->creator;
```

Iterating through the list of languages and showing their details can be done very easily with standard looping methods, such as `foreach`.

```
<?php
foreach ($languages->lang as $lang) {
    printf(
        "<p>%s appeared in %d and was created by %s.</p>",
        $lang["name"],
        $lang->appeared,
        $lang->creator
    );
}
```

Notice that I accessed the `lang` element's name attribute to retrieve the name of the language. You can access any attribute of an element represented as a `SimpleXMLElement` object using array notation like this.

Dealing With Namespaces

Many times you'll encounter namespaced elements while working with XML from different web services. Let's modify our `languages.xml` example to reflect the usage of namespaces:

```
<?xml version="1.0" encoding="utf-8"?>
<languages
 xmlns:dc="http://purl.org/dc/elements/1.1/ (http://purl.org/dc/elements/1.1/)">
 <lang name="C">
  <appeared>1972</appeared>
  <dc:creator>Dennis Ritchie</dc:creator>
```

```
  </lang>
  <lang name="PHP">
   <appeared>1995</appeared>
   <dc:creator>Rasmus Lerdorf</dc:creator>
  </lang>
  <lang name="Java">
   <appeared>1995</appeared>
   <dc:creator>James Gosling</dc:creator>
  </lang>
 </languages>
```

Now the `creator` element is placed under the namespace `dc` which points to *http://purl.org/dc/elements/1.1/*. If you try to print the creator of a language using our previous technique, it won't work. In order to read namespaced elements like this you need to use one of the following approaches.

The first approach is to use the namespace URI directly in your code when accessing namespaced elements. The following example demonstrates how:

```
<?php
$dc = $languages->lang[1]- >children("http://purl.org/dc/elements/1.1/ (http://purl
echo $dc->creator;
```

The `children()` method takes a namespace and returns the children of the element that are prefixed with it. It accepts two arguments; the first one is the XML namespace and the latter is an optional Boolean which defaults to false. If you pass true, the namespace will be treated as a prefix rather the actual namespace URI.

The second approach is to read the namespace URI from the document and use it while accessing namespaced elements. This is actually a cleaner way of accessing elements because you don't have to hardcode the URI.

```
<?php
$namespaces = $languages->getNamespaces(true);
$dc = $languages->lang[1]->children($namespaces["dc"]);

echo $dc->creator;
```

The `getNamespaces()` method returns an array of namespace prefixes with their associated URIs. It accepts an optional parameter which defaults to false. If you set it true then the method will return the namespaces used in parent and child nodes. Otherwise, it finds namespaces used within the parent node only.

Now you can iterate through the list of languages like so:

```php
<?php
$languages = simplexml_load_file("languages.xml");
$ns = $languages->getNamespaces(true);

foreach($languages->lang as $lang) {
    $dc = $lang->children($ns["dc"]);
    printf(
        "<p>%s appeared in %d and was created by %s.</p>",
        $lang["name"],
        $lang->appeared,
        $dc->creator
    );
}
```

A Practical Example – Parsing YouTube Video Feed

Let's walk through an example that retrieves the RSS feed from a YouTube channel displays links to all of the videos from it. For this we need to make a call to the following URL:

http://gdata.youtube.com/feeds/api/users//uploads

The URL returns a list of the latest videos from the given channel in XML format. We'll parse the XML and get the following pieces of information for each video:

- Video URL
- Thumbnail
- Title

We'll start out by retrieving and loading the XML:

```php
<?php
```

```
$channel = "channelName";
$url = "http://gdata.youtube.com/feeds/api/users/ (http://gdata.youtube.com/feeds/a
$xml = file_get_contents($url);

$feed = simplexml_load_string($xml);
$ns=$feed->getNameSpaces(true);
```

If you take a look at the XML feed you can see there are several `entity` elements each of which stores the details of a specific video from the channel. But we are concerned with only thumbnail image, video URL, and title. The three elements are children of `group`, which is a child of `entry`:

```
<entry>
   …
   <media:group>
      …
      <media:player url="video url"/>
      <media:thumbnail url="video url" height="height" width="width"/>
      <media:title type="plain">Title…</media:title>
      …
   </media:group>
   …
</entry>
```

We simply loop through all the `entry` elements, and for each one we can extract the relevant information. Note that `player`, `thumbnail`, and `title` are all under the `media` namespace. So, we need to proceed like the earlier example. We get the namespaces from the document and use the namespace while accessing the elements.

```
<?php
foreach ($feed->entry as $entry) {
    $group=$entry->children($ns["media"]);
    $group=$group->group;
    $thumbnail_attrs=$group->thumbnail[1]->attributes();
    $image=$thumbnail_attrs["url"];
    $player=$group->player->attributes();
    $link=$player["url"];
    $title=$group->title;
    printf('<p><a href="%s"><img src="%s" alt="%s"></a></p>',
```

```
11              $player, $image, $title);
12  }
```

Conclusion

Now that you know how to use SimpleXML to parse XML data, you can improve your skills by parsing different XML feeds from various APIs. But an important point to consider is that SimpleXML reads the entire DOM into memory, so if you are parsing large data sets then you may face memory issues. In those cases it’s advisable to use something other than SimpleXML, preferably an event-based parser such as XML Parser. To learn more about SimpleXML, check out its documentation (http://php.net/manual/en/book.simplexml.php).