http://www.cnblogs.com/likwo/archive/2011/08/24/2151793.html

XML处理是开发过程中经常遇到的，PHP对其也有很丰富的支持，本文只是对其中某几种解析技术做简要说明，包括：Xml parser, SimpleXML, XMLReader, DOMDocument。

**1。 XML Expat Parser：**

XML Parser使用Expat XML解析器。Expat是一种基于事件的解析器，它把XML文档视为一系列事件。当某个事件发生时，它调用一个指定的函数处理它。Expat是无验证的解析器，忽略任何链接到文档的DTD。但是，如果文档的形式不好，则会以一个错误消息结束。由于它基于事件，且无验证，Expat具有快速并适合web应用程序的特性。

XML Parser的优势是性能好，因为它不是将整个xml文档载入内存后再处理，而是边解析边处理。但也正因为如此，它不适合那些要对xml结构做动态调整、或基于xml上下文结构做复杂操作的需求。如果你只是要解析处理一个结构良好的xml文档，那么它可以很好的完成任务。需要注意的是XML Parser只支持三种编码格式：US-ASCII, ISO-8859-1和UTF-8，如果你的xml数据是其他编码，需要先转换成以上三个之一。  
XML Parser常用的解析方式大体有两种（其实就是两个函数）：xml\_parse\_into\_struct和xml\_set\_element\_handler。

xml\_parse\_into\_struct

此方法是将xml数据解析到两个数组中：  
index数组——包含指向Value 数组中值的位置的指针  
value数组——包含来自被解析的 XML 的数据

这俩数组文字描述起来有点麻烦，还是看个例子吧（来自php官方文档）

$simple = "<para><note>simple note</note></para>";

$p = xml\_parser\_create();

xml\_parse\_into\_struct($p, $simple, $vals, $index);

xml\_parser\_free($p);

echo "Index array\n";

print\_r($index);

echo "\nVals array\n";

print\_r($vals);

输出：  
Index array  
Array  
(  
    [PARA] => Array  
        (  
            [0] => 0  
            [1] => 2  
        )

    [NOTE] => Array  
        (  
            [0] => 1  
        )  
)

Vals array  
Array  
(  
    [0] => Array  
        (  
            [tag] => PARA  
            [type] => open  
            [level] => 1  
        )

    [1] => Array  
        (  
            [tag] => NOTE  
            [type] => complete  
            [level] => 2  
            [value] => simple note  
        )

    [2] => Array  
        (  
            [tag] => PARA  
            [type] => close  
            [level] => 1  
        )  
)

其中index数组以标签名为key，对应的值是一个数组，里面包括所有此标签在value数组中的位置。然后通过这个位置，找到此标签对应的值。

如果xml中每组数据格式有出入，不能做到完全统一，那么在写代码时要注意，说不定就得到了错误的结果。比如下面这个例子：

$xml = '

<infos>

<para><note>note1</note><extra>extra1</extra></para>

<para><note>note2</note></para>

<para><note>note3</note><extra>extra3</extra></para>

</infos>

';

$p = xml\_parser\_create();

xml\_parse\_into\_struct($p, $xml, $values, $tags);

xml\_parser\_free($p);

$result = array();

//下面的遍历方式有bug隐患

for ($i=0; $i<3; $i++) {

$result[$i] = array();

$result[$i]["note"] = $values[$tags["NOTE"][$i]]["value"];

$result[$i]["extra"] = $values[$tags["EXTRA"][$i]]["value"];

}

print\_r($result);

要是按照上面那种方式遍历，看似代码简单，但是暗藏危机，最致命的是得到错误的结果（extra3跑到第二个para里了）。所以要以一种比较严谨的方式遍历：

$result = array();

$paraTagIndexes = $tags['PARA'];

$paraCount = count($paraTagIndexes);

for($i = 0; $i < $paraCount; $i += 2) {

$para = array();

//遍历para标签对之间的所有值

for($j = $paraTagIndexes[$i]; $j < $paraTagIndexes[$i+1]; $j++) {

$value = $values[$j]['value'];

if(empty($value)) continue;

$tagname = strtolower($values[$j]['tag']);

if(in\_array($tagname, array('note','extra'))) {

$para[$tagname] = $value;

}

}

$result[] = $para;

}

其实我很少用xml\_parse\_into\_struct函数，所以上面所谓“严谨”的代码保不齐还会有其他情况下的bug。- -|  
xml\_set\_element\_handler

这种方式是为parser设置处理元素起始、元素终止的回调函数。配套的还有xml\_set\_character\_data\_handler用来为parser设置数据的回调函数。这种方式写的代码比较清晰，利于维护。

Example:

$xml = <<<XML

<infos>

<para><note>note1</note><extra>extra1</extra></para>

<para><note>note2</note></para>

<para><note>note3</note><extra>extra3</extra></para>

</infos>

XML;

$result = array();

$index = -1;

$currData;

function charactor($parser, $data) {

global $currData;

$currData = $data;

}

function startElement($parser, $name, $attribs) {

global $result, $index;

$name = strtolower($name);

if($name == 'para') {

$index++;

$result[$index] = array();

}

}

function endElement($parser, $name) {

global $result, $index, $currData;

$name = strtolower($name);

if($name == 'note' || $name == 'extra') {

$result[$index][$name] = $currData;

}

}

$xml\_parser = xml\_parser\_create();

xml\_set\_character\_data\_handler($xml\_parser, "charactor");

xml\_set\_element\_handler($xml\_parser, "startElement", "endElement");

if (!xml\_parse($xml\_parser, $xml)) {

echo "Error when parse xml: ";

echo xml\_error\_string(xml\_get\_error\_code($xml\_parser));

}

xml\_parser\_free($xml\_parser);

print\_r($result);

可见，set handler方式虽然代码行数多，但思路清晰，可读性更好，不过性能上略慢于第一种方式，而且灵活性不强。XML Parser支持PHP4，适用于于使用老版本的系统。对于PHP5环境，还是优先考虑下面的方法吧。

**2。 SimpleXML**

SimpleXML是PHP5后提供的一套简单易用的xml工具集，可以把xml转换成方便处理的对象，也可以组织生成xml数据。不过它不适用于包含namespace的xml，而且要保证xml格式完整(well-formed)。它提供了三个方法：simplexml\_import\_dom、simplexml\_load\_file、simplexml\_load\_string，函数名很直观地说明了函数的作用。三个函数都返回SimpleXMLElement对象，数据的读取/添加都是通过SimpleXMLElement操作。

$string = <<<XML

<?xml version='1.0'?>

<document>

<cmd>login</cmd>

<login>imdonkey</login>

</document>

XML;

$xml = simplexml\_load\_string($string);

print\_r($xml);

$login = $xml->login;//这里返回的依然是个SimpleXMLElement对象

print\_r($login);

$login = (string) $xml->login;//在做数据比较时，注意要先强制转换

print\_r($login);

SimpleXML的优点是开发简单，缺点是它会将整个xml载入内存后再进行处理，所以在解析超多内容的xml文档时可能会力不从心。如果是读取小文件，而且xml中也不包含namespace，那SimpleXML是很好的选择。

**3。 XMLReader**

XMLReader也是PHP5之后的扩展（5.1后默认安装），它就像游标一样在文档流中移动，并在每个节点处停下来，操作起来很灵活。它提供了对输入的快速和非缓存的流式访问，可以读取流或文档，使用户从中提取数据，并跳过对应用程序没有意义的记录。  
以一个利用google天气api获取信息的例子展示下XMLReader的使用，这里也只涉及到一小部分函数，更多还请参考官方文档。

$xml\_uri = 'http://www.google.com/ig/api?weather=Beijing&hl=zh-cn';

$current = array();

$forecast = array();

$reader = new XMLReader();

$reader->open($xml\_uri, 'gbk');

while ($reader->read()) {

//get current data

if ($reader->name == "current\_conditions" && $reader->nodeType == XMLReader::ELEMENT) {

while($reader->read() && $reader->name != "current\_conditions") {

$name = $reader->name;

$value = $reader->getAttribute('data');

$current[$name] = $value;

}

}

//get forecast data

if ($reader->name == "forecast\_conditions" && $reader->nodeType == XMLReader::ELEMENT) {

$sub\_forecast = array();

while($reader->read() && $reader->name != "forecast\_conditions") {

$name = $reader->name;

$value = $reader->getAttribute('data');

$sub\_forecast[$name] = $value;

}

$forecast[] = $sub\_forecast;

}

}

$reader->close();

XMLReader和XML Parser类似，都是边读边操作，较大的差异在于SAX模型是一个“推送”模型，其中分析器将事件推到应用程序，在每次读取新节点时通知应用程序，而使用XmlReader的应用程序可以随意从读取器提取节点，可控性更好。  
由于XMLReader基于libxml，所以有些函数要参考文档看看是否适用于你的libxml版本。

**4。 DOMDocument**

DOMDocument还是PHP5后推出的DOM扩展的一部分，可用来建立或解析html/xml，目前只支持utf-8编码。

$xmlstring = <<<XML

<?xml version='1.0'?>

<document>

<cmd attr='default'>login</cmd>

<login>imdonkey</login>

</document>

XML;

$dom = new DOMDocument();

$dom->loadXML($xmlstring);

print\_r(getArray($dom->documentElement));

function getArray($node) {

$array = false;

if ($node->hasAttributes()) {

foreach ($node->attributes as $attr) {

$array[$attr->nodeName] = $attr->nodeValue;

}

}

if ($node->hasChildNodes()) {

if ($node->childNodes->length == 1) {

$array[$node->firstChild->nodeName] = getArray($node->firstChild);

} else {

foreach ($node->childNodes as $childNode) {

if ($childNode->nodeType != XML\_TEXT\_NODE) {

$array[$childNode->nodeName][] = getArray($childNode);

}

}

}

} else {

return $node->nodeValue;

}

return $array;

}

从函数名上看感觉跟JavaScript很像，应该是借鉴了一些吧。DOMDocument也是一次性将xml载入内存，所以内存问题同样需要注意。PHP提供了这么多的xml处理方式，开发人员在选择上就要花些时间了解，选择适合项目需求及系统环境、又便于维护的方法。

转

<http://www.phpzh.com/archives/525/2>