第 6 章　PHP 与 XML 格式操作

XML 是一种流行的半结构化文件格式，以一种类似数据库的格式存储数据。在实际应用中，一些简单的、安全性较低的数据往往使用 XML 文件的格式进行存储。这样做的好处一方面可以通过减少与数据库的交互性操作提高读取效率，另一方面可以有效利用 XML 的优越性降低程序的编写难度。

PHP 提供了一整套的读取 XML 文件的方法，很容易的就可以编写基于 XML 的脚本程序。本章将要介绍 PHP 与 XML 的操作方法，并对几个常用的 XML 类库做一些简要介绍。

6.1　XML 简介

XML 是“可扩展性标识语言（eXtensible Markup Language）”的缩写，是一种类似于 HTML 的标记性语言。但是与 HTML 不同，XML 主要用于描述数据和存放数据，而 HTML 主要用于显示数据。

XML 是一种“元标记”语言，开发者可以根据自己的需要创建标记的名称。例如，下面的 XML 代码可以用来描述一条留言。

```
<thread>
    <title>Welcome</title>
    <author>Simon</author>
    <content>Welcome to XML guestbook!!</content>
</thread>
```

其中，<thread>与</thread>标签标记了这是一段留言。在留言中有标题、作者、内容，完整的表述了一条留言信息。

在一个 XML 文件的顶部，通常使用<?xml version="1.0"?>来标识 XML 数据的开始和 XML 数据使用标准的版本信息。在浏览器中访问 XML 文件可以看到层次分明的 XML 数据信息，如图 6-1 所示。

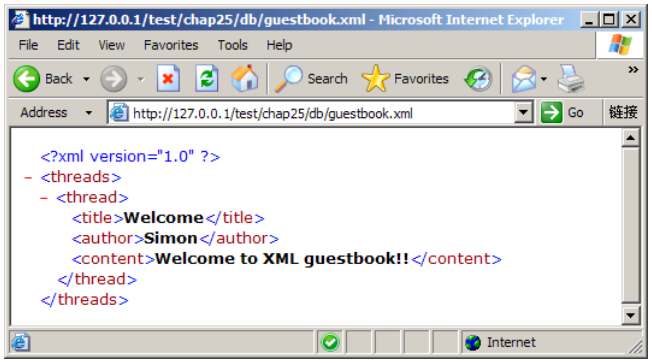

图 6-1　一个 XML 示例

XML 的发展非常迅速，近些年来很多软件开发商都开始采用 XML 的开发标准进行应用程序的开发。并且，很多新兴技术都架构在 XML 数据之上。这意味着 XML 将与 HTML 一样成为 Web 技术不可或缺的一部分。

6.2　简单的 XML 操作

在实际应用中，PHP 与 XML 的交互操作应用非常广泛。SimpleXML 组件是 PHP5 新增加的一个简单的 XML 操作组件，与传统的 XML 组件相比，SimpleXML 组件的使用非常简单。本节将对使用 SimpleXML 组件操作 XML 的方法做一下详细介绍。

6.2.1　创建一个 SimpleXML 对象

SimpleXML 对象是用来临时存储 XML 数据的临时变量，对 XML 进行的操作都是通过操作 SimpleXML 对象来完成的。SimpleXML 组件提供了两种创建 SimpleXML 对象的方法。

第一种方法是使用 simplexml_load_string 函数读取一个字符串型变量中的 XML 数据来完成创建的，其语法格式如下所示。

```
simplexml_load_string(string data)
```

这里的 data 变量用于存储 XML 数据。以下代码使用 simplexml_load_string 函数创建了一个 SimpleXML 对象。

```
<?php
$data = <<<XML                                          //定义 XML 数据
<?xml version='1.0'?>
<departs>
<depart>
    <name>production support</name>
    <employees>
        <employee>
            <serial_no>100001</serial_no>
            <name>Simon</name>
            <age>24</age>
            <birthday>1982-11-06</birthday>
            <salary>5000.00</salary>
            <bonus>1000.00</bonus>
        </employee>
        <employee>
            <serial_no>100002</serial_no>
            <name>Elaine</name>
            <age>24</age>
            <birthday>1982-01-01</birthday>
            <salary>6000.00</salary>
            <bonus>2000.00</bonus>
        </employee>
    </employees>
</depart>
<depart>
    <name>testing center</name>
    <employees>
        <employee>
            <serial_no>110001</serial_no>
            <name>Helen</name>
```

```
                <age>23</age>
                <birthday>1983-07-21</birthday>
                <salary>5000.00</salary>
                <bonus>1000.00</bonus>
            </employee>
        </employees>
</depart>
</departs>
XML;
$xml = simplexml_load_string($data);                    //创建 SimpleXML 对象
print_r($xml);                                          //输出 XML
?>
```

在上面的例子中，$data 变量存储了一段 XML 数据。simplexml_load_string 函数将变量$data 转化成 SimpleXML 对象。通过 print_r 函数的输出可以看出该对象的结构，运行结果如下所示。

```
SimpleXMLElement Object
(
    [depart] => Array
        (
            [0] => SimpleXMLElement Object
                (
                    [name] => production support
                    [employees] => SimpleXMLElement Object
                        (
                            [employee] => Array
                                (
                                    [0] => SimpleXMLElement Object
                                        (
                                            [serial_no] => 100001
                                            [name] => Simon
                                            [age] => 24
                                            [birthday] => 1982-11-06
                                            [salary] => 5000.00
                                            [bonus] => 1000.00
                                        )

                                    [1] => SimpleXMLElement Object
                                        (
                                            [serial_no] => 100002
                                            [name] => Elaine
                                            [age] => 24
                                            [birthday] => 1982-01-01
                                            [salary] => 6000.00
                                            [bonus] => 2000.00
                                        )

                                )

                        )

                )
```

```
            [1] => SimpleXMLElement Object
                (
                    [name] => testing center
                    [employees] => SimpleXMLElement Object
                        (
                            [employee] => SimpleXMLElement Object
                                (
                                    [serial_no] => 110001
                                    [name] => Helen
                                    [age] => 23
                                    [birthday] => 1983-07-21
                                    [salary] => 5000.00
                                    [bonus] => 1000.00
                                )

                        )

                )

        )

)
```

从输出结果可以看出，SimpleXML 对象的结构与 XML 数据的格式完全相同。

第二种方法是使用 simplexml_load_flie 函数读取一个 XML 文件来完成创建的，其语法格式如下所示。

```
simplexml_load_file(string filename)
```

这里的 filename 变量是用于存储 XML 数据文件的文件名及其所在路径。以下代码使用 simplexml_load_file 函数来创建了一个 SimpleXML 对象。

```
<?php
$xml = simplexml_load_file('example.xml');                    //创建 SimpleXML 对象
print_r($xml);                                                //输出 XML
?>
```

其中，example.xml 存储的数据与上面的$data 完全相同，运行结果也与上面完全相同。

上面两种方法实现了同样的功能，其区别就在于 XML 的数据源不同。如果 XML 的数据源在 PHP 脚本文件中，则需要使用 simplexml_load_string 来进行创建。如果 XML 的数据源在一个单独的 XML 文件中，则需要使用 simplexml_load_file 来进行创建。

6.2.2　读取 SimpleXML 对象中的 XML 数据

前面介绍了使用 print_r 函数来读取 SimpleXML 对象中的数据，其返回结果与数组的结构类似。显然，这种显示方式在实际应用中是不可取的。在这里将介绍其他的几种读取 SimpleXML 对象中 XML 数据的方法。

1．var_dump 函数显示对象详细信息

var_dump 函数可以用于显示 SimpleXML 对象的详细信息，与 print_r 函数相比，var_dump 函数显示的信息更为完整，其语法如下所示。

```
void var_dump(object1, object2 … )
```

以下代码使用 var_dump 函数输出了上面例子中对象的详细信息。

```
<?php
$xml = simplexml_load_file('example.xml');                    //创建 SimpleXML 对象
var_dump($xml);                                               //输出 XML
?>
```

运行结果如下所示。

```
object(SimpleXMLElement)#1 (1) {
  ["depart"]=>
  array(2) {
    [0]=>
    object(SimpleXMLElement)#2 (2) {
      ["name"]=>
      string(18) "production support"
      ["employees"]=>
      object(SimpleXMLElement)#4 (1) {
        ["employee"]=>
        array(2) {
          [0]=>
          object(SimpleXMLElement)#5 (6) {
            ["serial_no"]=>
            string(6) "100001"
            ["name"]=>
            string(5) "Simon"
            ["age"]=>
            string(2) "24"
            ["birthday"]=>
            string(10) "1982-11-06"
            ["salary"]=>
            string(7) "5000.00"
            ["bonus"]=>
            string(7) "1000.00"
          }
          [1]=>
          object(SimpleXMLElement)#6 (6) {
            ["serial_no"]=>
            string(6) "100002"
            ["name"]=>
            string(6) "Elaine"
            ["age"]=>
            string(2) "24"
            ["birthday"]=>
            string(10) "1982-01-01"
            ["salary"]=>
            string(7) "6000.00"
            ["bonus"]=>
            string(7) "2000.00"
          }
        }
      }
```

```
    }
    [1]=>
    object(SimpleXMLElement)#3 (2) {
      ["name"]=>
      string(14) "testing center"
      ["employees"]=>
      object(SimpleXMLElement)#7 (1) {
        ["employee"]=>
        object(SimpleXMLElement)#8 (6) {
          ["serial_no"]=>
          string(6) "110001"
          ["name"]=>
          string(5) "Helen"
          ["age"]=>
          string(2) "23"
          ["birthday"]=>
          string(10) "1983-07-21"
          ["salary"]=>
          string(7) "5000.00"
          ["bonus"]=>
          string(7) "1000.00"
        }
      }
    }
  }
}
```

与前面 print_r 输出的结果相比较，var_dump 函数的输出结果的结构更为严谨，并且将对象中的每一个属性的数据类型均作出分析。在实际应用中，var_dump 函数往往用于程序调试时的对象检测。

2．读取 XML 数据中的标签

与操作数组类型的变量类似，读取 XML 也可以通过类似的方法来完成。例如，如果需要读取上面 XML 数据中每一个“depart”标签下的“name”属性，可以通过使用 foreach 函数来完成，如以下代码所示。

```
<?php
$xml = simplexml_load_file('example.xml');                    //读取 XML 文件
foreach($xml->depart as $a)                                   //循环读取 XML 数据中的每一个 depart 标签
{
    echo "$a->name <BR>";                                     //输出其中的 name 属性
}
?>
```

运行结果如下所示。

```
production support
testing center
```

也可以使用方括号“[]”来直接读取 XML 数据中指定的标签。以下代码输出了上面 XML 数据中的第一个“depart”标签的“name”属性。

```
<?php
$xml = simplexml_load_file('example.xml');                    //读取 XML 文件
echo $xml->depart->name[0];                                   //输出节点
?>
```

运行结果如下所示。

```
production support
```

对于一个标签下的所有子标签，SimpleXML 组件提供了 children 方法进行读取。例如，对于上面的 XML 数据中的“depart”标签，其下包括两个子标签：“name”和“employees”。以下代码实现了对第一个“depart”标签下的子标签的读取。

```
<?php
$xml = simplexml_load_file('example.xml');
foreach ($xml->depart->children() as $depart)              //循环读取 depart 标签下的子标签
{
    var_dump($depart);                                     //输出标签的 XML 数据
}
?>
```

运行结果如下所示。

```
object(SimpleXMLElement)#3 (1) {
  [0]=>
  string(18) "production support"
}
object(SimpleXMLElement)#5 (1) {
  ["employee"]=>
  array(2) {
    [0]=>
    object(SimpleXMLElement)#3 (6) {
      ["serial_no"]=>
      string(6) "100001"
      ["name"]=>
      string(5) "Simon"
      ["age"]=>
      string(2) "24"
      ["birthday"]=>
      string(10) "1982-11-06"
      ["salary"]=>
      string(7) "5000.00"
      ["bonus"]=>
      string(7) "1000.00"
    }
    [1]=>
    object(SimpleXMLElement)#6 (6) {
      ["serial_no"]=>
      string(6) "100002"
      ["name"]=>
      string(6) "Elaine"
      ["age"]=>
      string(2) "24"
      ["birthday"]=>
      string(10) "1982-01-01"
      ["salary"]=>
      string(7) "6000.00"
      ["bonus"]=>
      string(7) "2000.00"
```

```
    }
  }
}
```

可以看出，使用 children 方法后，所有的子标签均被当作一个新的 XML 文件进行处理。

3．基于 XML 数据路径的查询

SimpleXML 组件提供了一种基于 XML 数据路径的查询方法。XML 数据路径即从 XML 的根到某一个标签所经过的全部标签。这种路径使用斜线“/”隔开标签名。例如，对于上面的 XML 数据，要查询所有的标签“name”中的值，从根开始要经过 departs、depart、employees 和 employee 标签，则其路径为“/departs/depart/employees/employee/name”。

SimpleXML 组件使用 xpath 方法来解析路径，其语法格式如下所示。

```
xpath(string path)
```

其中的 path 为路径。该方法返回了一个包含有所有要查询标签值的数组。以下代码查询了上面 XML 数据中的所有 name 标签。

```
<?php
$xml = simplexml_load_file('example.xml');                          //读取 XML 文件
$result = $xml->xpath('/departs/depart/employees/employee/name');   //定义节点
var_dump($result);                                                  //输出节点
?>
```

运行结果如下所示。

```
array(3) {
  [0]=>
  object(SimpleXMLElement)#2 (1) {
    [0]=>
    string(5) "Simon"
  }
  [1]=>
  object(SimpleXMLElement)#3 (1) {
    [0]=>
    string(6) "Elaine"
  }
  [2]=>
  object(SimpleXMLElement)#4 (1) {
    [0]=>
    string(5) "Helen"
  }
}
```

可以看出，所有的 name 标签均被查询出来。

6.2.3　XML 数据的修改

对于 XML 数据的修改与读取 XML 数据中的标签方法类似。即通过直接修改 SimpleXML 对象中的标签的值来实现。以下代码实现了对上面 XML 数据中第一个“depart”标签的“name”子标签的修改。

```
<?php
$xml = simplexml_load_file('example.xml');                  //读取 XML
$xml->depart->name[0] = "Human Resource";                   //修改节点
?>
```

修改后，并不会对 XML 文件有任何影响。但是，在程序中，对于 SimpleXML 对象的读取将使用修改过的值。

6.2.4　标准化 XML 数据

SimpleXML 还提供了一种标准化 XML 数据的方法 asXML。asXML 方法可以有效的将 SimpleXML 对象中的内容按照 XML 1.0 标准进行重新编排并以字符串的数据类型返回。以下代码实现了对上面 XML 数据的标准化。

```
<?php
$xml = simplexml_load_file('example.xml');                    //读取 XML 数据
echo $xml->asXML();                                           //标准化 XML 数据
?>
```

6.2.5　XML 数据的存储

将 SimpleXML 对象中的 XML 数据存储到一个 XML 文件的方法非常简单，即将 asXML 方法的返回结果输出到一个文件中即可。以下代码首先将 XML 文件中的 depart name 进行了修改，然后将修改过的 XML 数据输出到另一个 XML 文件。

```
<?php
$xml = simplexml_load_file('example.xml');                    //读取 XML 数据
$newxml = $xml->asXML();                                      //标准化 XML 数据
$fp = fopen("newxml.xml", "w");                               //打开要写入 XML 数据的文件
fwrite($fp, $newxml);                                         //写入 XML 数据
fclose($fp);                                                  //关闭文件
?>
```

代码运行后，可以看到在 newxml.xml 文件中的 XML 数据如下所示。

```
<?xml version="1.0"?>
<departs>
<depart>
    <name>Human Resource</name>
    <employees>
        <employee>
            <serial_no>100001</serial_no>
            <name>Simon</name>
            <age>24</age>
            <birthday>1982-11-06</birthday>
            <salary>5000.00</salary>
            <bonus>1000.00</bonus>
        </employee>
        <employee>
            <serial_no>100002</serial_no>
            <name>Elaine</name>
            <age>24</age>
            <birthday>1982-01-01</birthday>
            <salary>6000.00</salary>
            <bonus>2000.00</bonus>
        </employee>
```

```
    </employees>
</depart>
<depart>
    <name>testing center</name>
    <employees>
        <employee>
            <serial_no>110001</serial_no>
            <name>Helen</name>
            <age>23</age>
            <birthday>1983-07-21</birthday>
            <salary>5000.00</salary>
            <bonus>1000.00</bonus>
        </employee>
    </employees>
</depart>
</departs>
```

可以看出，对于 XML 文件的修改已经保存到输出文件中了。

6.3　XML 文档的动态创建

在实际应用中，时而会需要动态生成 XML 文档的操作。前面介绍的 SimpleXML 组件并不提供创建 XML 文档的方法。因此，如果需要动态创建 XML 文档，往往使用 DOM 组件进行创建。DOM 是文档对象模型 Document Object Model 的缩写，DOM 组件提供了对 XML 文档的树型解析模式。以下代码使用 DOM 组件创建了一个 XML 文档。

```
<?php
//创建一个新的 DOM 文档
$dom = new DomDocument();
//在根节点创建 departs 标签
$departs = $dom->createElement('departs');
$dom->appendChild($departs);
//在 departs 标签下创建 depart 子标签
$depart = $dom->createElement('depart');
$departs->appendChild($depart);
//在 depart 标签下创建 employees 子标签
$employees = $dom->createElement('employees');
$depart->appendChild($employees);
//在 employees 标签下创建 employee 子标签
$employee = $dom->createElement('employee');
$employees->appendChild($employee);
//在 employee 标签下创建 serial_no 子标签
$serial_no = $dom->createElement('serial_no');
$employee->appendChild($serial_no);
//为 serial_no 标签添加值节点 100001
$serial_no_value = $dom->createTextNode('100001');
$serial_no->appendChild($serial_no_value);
//输出 XML 数据
echo $dom->saveXML();
```

```
?>
```

输出结果如下所示。

```
<?xml version="1.0"?>
<departs>
<depart>
<employees>
<employee>
<serial_no>100001</serial_no>
</employee>
</employees>
</depart>
</departs>
```

DOM 组件除了可以用来动态创建 XML 文档外，还可以用来读取 XML 文件。以下代码实现了对前面 XML 文件的读取。

```
<?php
   $dom = new DomDocument();                                  //创建 DOM 对象
   $dom->load('example.xml');                                 //读取 XML 文件
   $root = $dom->documentElement;                             //获取 XML 数据的根
   read_child($root);                                         //调用 read_child 函数读取根对象

   function read_child($node)
   {
         $children = $node->childNodes;                       //获得$node 的所有子节点

         foreach($children as $e)                             //循环读取每一个子节点
         {
               if($e->nodeType == XML_TEXT_NODE)              //如果子节点为文本型则输出
               {
                     echo $e->nodeValue."<BR>";
               }
               else if($e->nodeType == XML_ELEMENT_NODE)      //如果子节点为节点对象，则调用函数处理
               {
                     read_child($e);
               }
         }
    }
?>
```

运行结果如下所示。

```
production support
100001
Simon
24
1982-11-06
5000.00
1000.00
100002
Elaine
24
1982-01-01
```

```
6000.00
2000.00
testing center
110001
Helen
23
1983-07-21
5000.00
1000.00
```

上面的例子使用了递归的方式对 XML 数据进行了处理，实现了输出 XML 数据中的所有文本型标签的功能。

6.4　XML 应用实例——留言本

前面介绍了 XML 的基本操作，本节将以设计一个 XML 留言本为例来详细说明在实际应用中如何实现 PHP 与 XML 数据的交互操作。

6.4.1　XML 文件结构设计

XML 文件用于存储 XML 数据，也就是留言本中的留言。这里，对于每条留言，在 XML 数据中主要包括三项内容：留言标题、留言者姓名、留言内容。因此，将 XML 文件的格式设计如下。

```
<?xml version="1.0"?>
<threads>
<thread>
      <title>这里是留言的标题</title>
      <author>这里是留言者</author>
      <content>这里是留言内容</content>
</thread>
</threads>
```

6.4.2　提交页面的编写

提交留言页面由两个页面组成。一个是让访问者用来书写留言的表单的 HTML 文件，一个是用来处理访问者输入的 PHP 脚本。

表单的 HTML 代码如下所示。

```
<html>
<head>
<title>发表新的留言</title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body>
<H1><p align="center">发表新的留言</p></H1>
<form name="form1" method="post" action="Post.php">
  <table width="500" border="0" align="center" cellpadding="0" cellspacing="0">
    <tr>
```

```
        <td>标题</td>
        <td><input name="title" type="text" id="title" size="50"></td>
      </tr>
      <tr>
        <td>作者</td>
        <td><input name="author" type="text" id="author" size="20"></td>
      </tr>
      <tr>
        <td>内容</td>
        <td><textarea name="content" cols="50" rows="10" id="content"></textarea></td>
      </tr>
  </table>
  <p align="center">
    <input type="submit" value="Submit">
    <input type="reset" value="Reset">
</p>
</form>
</body>
</html>
```

对于用来处理用户输入的 PHP 脚本，其基本逻辑是首先创建一个 DOM 对象，然后读取 XML 文件中的 XML 数据，接下来在 XML 对象上创建新的节点并将用户的输入储存起来，最后将 XML 数据输出到原来的 XML 文件中。具体实现代码如下所示。

```
<?php
$guestbook = new DomDocument();                                    //创建一个新的 DOM 对象
$guestbook->load('DB/guestbook.xml');                              //读取 XML 数据
$threads = $guestbook->documentElement;                            //获得 XML 结构的根
//创建一个新 thread 节点
$thread = $guestbook->createElement('thread');
$threads->appendChild($thread);
//在新的 thread 节点上创建 title 标签
$title = $guestbook->createElement('title');
$title->appendChild($guestbook->createTextNode($_POST['title']));
$thread->appendChild($title);
//在新的 thread 节点上创建 author 标签
$author = $guestbook->createElement('author');
$author->appendChild($guestbook->createTextNode($_POST['author']));
$thread->appendChild($author);
//在新的 thread 节点上创建 content 标签
$content = $guestbook->createElement('content');
$content->appendChild($guestbook->createTextNode($_POST['content']));
$thread->appendChild($content);
//将 XML 数据写入文件
$fp = fopen("DB/guestbook.xml", "w");
if(fwrite($fp, $guestbook->saveXML()))
      echo "留言提交成功";
else
      echo "留言提交失败";
fclose($fp);
?>
```

在浏览器中运行上述 HTML 文件并填写适当的留言内容，如图 6-2 所示。

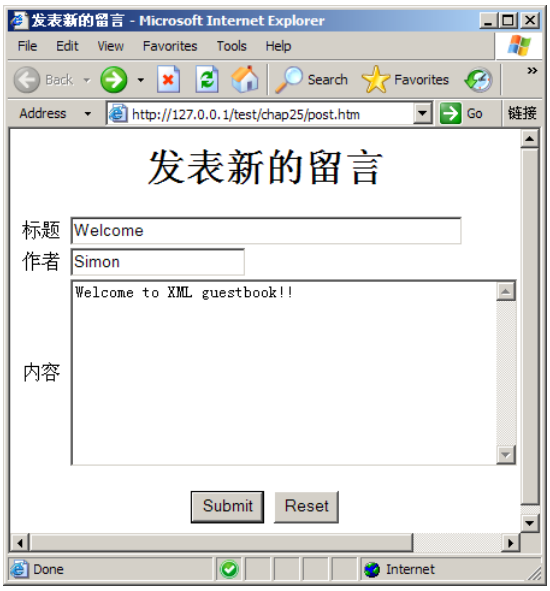

图 6-2　发表新留言界面

单击【Submit】按钮后，XML 文件中的内容如下所示。

```
<?xml version="1.0"?>
<threads>
<thread>
        <title>Welcome</title>
        <author>Simon</author>
        <content>Welcome to XML guestbook!!</content>
</thread>
</threads>
```

可以看到 XML 文件中已经将留言存储起来了。

6.4.3　显示页面的编写

显示页面可以使用前面介绍的 SimpleXML 组件很容易的实现，具体实现代码如下所示。

```
<?php
//打开用于存储留言的 XML 文件
$guestbook = simplexml_load_file('DB/guestbook.xml');

foreach($guestbook->thread as $th)                    //循环读取 XML 数据中的每一个 thread 标签
{
        echo "<B>标题：</B>".$th->title."<BR>";
        echo "<B>作者：</B>".$th->author."<BR>";
        echo "<B>内容：</B><PRE>".$th->content."</PRE>";
        echo "<HR>";
}
?>
```

在浏览器中查看运行结果如图 6-3 所示。

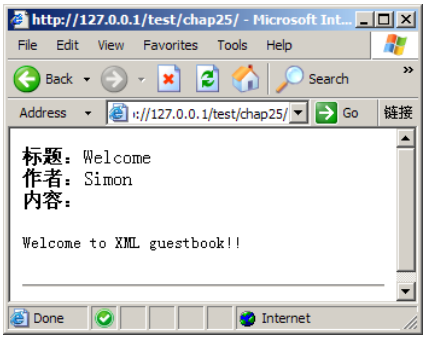

图 6-3　查看留言界面

可以看到在上一小节中添加的留言被显示出来了。

6.5　小结

本章介绍了 PHP 与 XML 数据的常用交互操作。随着 XML 技术的日益发展与成熟，实际应用中的 XML 技术的使用也越来越广泛。例如，时下流行的 RSS 订阅信息就是 XML 技术的一项重要应用。因此，对于 PHP 与 XML 数据的常用交互操作也是 PHP 技术的一个重要组成部分。

本章介绍的使用 PHP 创建 XML 文件是本章的难点。读者可以通过调试本章的 XML 留言本实例来加深对 XML 数据操作的理解。