*Лекция 14: Взаимодействие PHP и XML*

### **Введение**

Прежде чем начать изучать способы работы *PHP* с *XML*, полезно хотя бы вкратце ознакомиться с самой технологией *XML*. В первой части лекции будут рассмотрены основные понятия этой технологии, базовый *синтаксис* XML–документов, области ее применения, а также понятие *DOM XML*. Вторая часть лекции посвящена непосредственно вопросам обработки *XML* при помощи *PHP*. Сюда входит *установка расширения DOM XML*, описание и примеры использования некоторых встроенных в *PHP* функций для обработки XML-документов.

В качестве примера рассмотрим *XML* -*файл*, содержащий описания личностей, и попытаемся научиться добавлять, удалять и находить *личность* или отдельные *элементы* ее описания в этом файле с помощью *PHP*.

### **XML**

#### **Основные понятия**

***XML (Extensible Markup Language)*** – это расширяемый язык разметки, являющийся подмножеством языка *SGML* и поэтому имеющий общие с ним цели – разметка любого типа документов.

*XML* имеет много общего с языком разметки HTML, но они различны по своему назначению. HTML, как и *XML*, является подмножеством *SGML*, но предназначен только для отображения информации в браузере. *XML* же предназначен для структурированного *хранения информации* и не содержит никаких конструкций для отображения этих данных.

Для отображения данных, хранящихся в *XML* -формате, используются различного рода преобразователи, например язык трансформаций *XSLT*. Комбинация *XML* + преобразователь позволяет достичь того же результата, что и использование HTML-форматированного отображения документа пользователю. Однако в случае использования *XML* и преобразователя данные хранятся отдельно от их представления, т. е. от инструкций о том, как они должны отображаться.

*XML* очень похож на HTML по своему синтаксису, за исключением того, что теги *XML* можно придумывать собственные. По аналогии с HTML внутри *XML* -тегов можно использовать атрибуты. Каждый тег *XML* должен иметь соответствующий *закрывающий тег* (в HTML это не является обязательным). Каждый XML-документ начинается со строки декларации, например:

<?xml version='1.0' encoding='UTF-16'

standalone='yes' ?>

которая указывает на версию языка *XML* (атрибут version), кодировку текста в этом документе (атрибут encoding ) и показывает, существует ли документ сам по себе или зависит от других файлов (атрибут standalone ).

**Приведем пример XML-документа.**

Пусть у нас имеется письмо. Содержащуюся в нем информацию можно хранить в следующем виде:

<?xml version='1.0' encoding='KOI8-R'

standalone='yes' ?>

<note title="Письмо">

<to>Маша Петрова</to>

<body>Привет, Маша!

Как у тебя дела?

У меня все хорошо.

Собираюсь приехать

к тебе в гости.</body>

<from>Вася Иванов</from>

</note>

Сохранив этот файл, например, как note.xml, мы можем переслать его кому-нибудь, просмотреть с помощью браузера или использовать в программе. В частности, можно использовать этот XML-документ в своей PHP-программе.

С помощью *XML* можно создать документ для описания любой информации. Для того чтобы другие разработчики могли создавать документы в этом же *XML* -языке (т.е. используя те же теги) и чтобы можно было проверить правильность созданного документа (его соответствие выбранному *XML* -языку), нужно формально описать используемый *XML* -язык. Это делается с помощью механизмов *DTD (Document Type Definition)* или *Schemas*, которые дают возможность создавать шаблоны для новых типов документов. Для экспериментов в домашних условиях создавать *DTD* -описания необязательно. Важно понимать, что *XML* на самом деле вовсе не язык, а стандарт для создания языков, отвечающих критериям *XML*.

Между XML-документами могут быть установлены ссылки, один документ может включать в себя другой, т.е. *структура документа*может быть сколь угодно сложной.

*Элементы* в *XML* -документах могут быть связаны между собой *отношениями родитель/потомок* или родственник/родственник. В нашем примере <note> является *родителем* <to>, который в свою очередь является *потомком* <note>, а <to>, <body> и <from>являются родственниками. Собственно текст тоже является *потомком**элемента*, например, " Вася Иванов " – *потомок**элемента*<from>. Такая структура называется деревом; любые части дерева, имеющие *потомков*, называются ветвями, а не имеющие – листьями.

#### **Объектная модель XML-документа**

*Структура XML-документа* очень напоминает *объектную модель*: она иерархична, одни *элементы* могут быть *потомками* других. Любой *XML* -документ можно представить в качестве *объектной модели*. При этом одни *элементы* (например, теги) становятся объектами, другие (атрибуты и текстовые *элементы* ) – их свойствами. Например, для документа

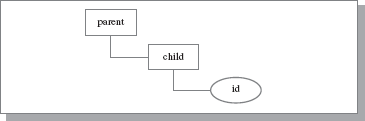
<parent>

<child id="text">Добрый день!</child>

</parent>

*объектная модель* может выглядеть как показано на [рис. 14.1](https://www.intuit.ru/studies/courses/42/42/lecture/27201?page=2#image.14.1)

Прямоугольниками изображены объекты, овалом – свойства объектов.

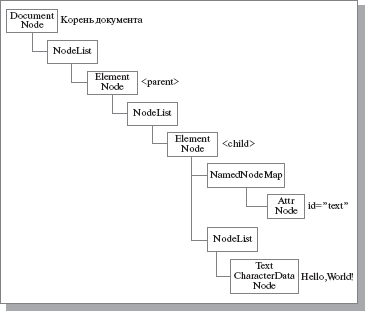


**Рис. 14.1.**Пример объектной модели

Для каждого конкретного *XML* -документа можно создать свою *объектную модель*, но есть более общая *объектная модель*, применимая для любого *XML* -документа, вне зависимости от его *структуры*. Эта модель называется *DOM (Document Object Model)*.

Обычно *DOM* добавляется как слой между *XML* -*парсером* и приложением, которому требуется информация из документа. То есть *парсер* берет данные из документа и передает их в *DOM*. Затем *DOM* используется приложениями более высокого уровня.

Хотя *DOM* расшифровывается как *"объектная модель документа"*, работает *DOM* преимущественно с интерфейсами, а не с объектами. Интерфейс – это соглашение, по которому поддерживаются определенные свойства и методы, применяемые к некоему объекту. На [рис. 14.2](https://www.intuit.ru/studies/courses/42/42/lecture/27201?page=2#image.14.2) показано, как приведенный выше *XML* -документ представляется в *DOM*:



**Рис. 14.2.**Пример DOM (Document Object Model)

Каждый прямоугольник представляет собой объект, имена в прямоугольниках соответствуют интерфейсам, которые будут реализованы каждым объектом. Каждый объект может реализовывать несколько подходящих интерфейсов. Например, объект представляющий символьные данные " Hello,World!", реализует интерфейсы Text, CharacterData, Node.

Преимущество модели *DOM* состоит в том, что она универсальна, т.е. может применяться для документов любого типа. Ее использование может существенно упростить обработку *XML* -документов.

Но вернемся к технологии *XML*. Для чего она используется? В основном для *хранения и передачи данных*. В последнее время *хранение данных в виде набора XML-файлов* рассматривается даже как альтернатива реляционным базам данных. Но наиболее прогрессивные разработчики стремятся задействовать совместно *XML* и базы данных, пользуясь достоинствами обеих технологий. В частности, *XML*удобнее использовать для *передачи данных*, а базу данных – для *их хранения* и обработки.

Например, главная компания и ее филиал в другом городе имеют два разных сайта поддержки, написанных на PHP. На сайте филиала нужно сообщать обо всех основных событиях, происходящих в главной компании, и наоборот. Поэтому обе организации ежедневно обмениваются новостями в универсальном формате *XML*, а эти файлы создаются и обрабатываются *PHP-скриптами* и отображаются на обоих сайтах.

### **Взаимодействие PHP и XML**

#### **Расширения SAX и DOM XML**

Для работы с *XML* -документами можно использовать язык PHP. В **PHP** для этого существует два модуля, реализующие два разных стандарта обработки *XML* -данных: *SAX (Simple API for XML)* и *DOM (Document Object Model)*.

Стандарт *SAX* ([http://www.saxproject.org](http://www.saxproject.org/)) не является стандартом W3C и описывает метод обработки *XML* -документов для получения из них данных. То есть этот метод обработки *XML* -документов позволит только прочитать данные из *XML* -документа, и не более того. Создавать и изменять *XML* -документы с его помощью невозможно.

***SAX*** основан на так называемом *событийном программировании*. Его особенность заключается в том, что вы предоставляете парсеру *XML* набор собственных функций, которые будут заниматься обработкой различных типов *XML* -данных ( *элементов* (тегов), текста и т.п.), а *парсер* затем будет сам вызывать ваши функции в процессе обработки *XML* -документа, передавая им найденные данные. Функции будут вызываться в той же последовательности, в которой соответствующие данные располагаются в *XML* -документе.

Другим стандартом для обработки *XML* -данных является *DOM* – стандарт **W3C**, спецификацию которого можно найти на сайте консорциума (<http://www.w3c.org/DOM>). В отличие от *SAX*, этот метод позволяет производить любые операции с *XML* -данными в достаточно удобной форме – представляя *XML* -документ как дерево объектов.

Модуль, реализующий этот стандарт, называется *DOM XML*. Он не входит в основной набор модулей PHP, но может быть установлен как расширение. API этого модуля старается как можно более точно следовать стандарту *DOM* level 2. Кроме того, существует множество дополнительных функций. Эти функции включены для совместимости с предыдущими версиями расширения, и использовать их в новых скриптах не рекомендуется. Кроме того, у расширения *DOM XML* есть проблемы с русской кодировкой. *Парсер* обрабатывает текст только в кодировке UTF-8, поэтому текст нужно каждый раз перекодировать с помощью функции iconv. Отсюда и необходимость установки расширения iconv вместе с *DOM XML*.

Модуль *DOM XML* является мощным и удобным в использовании средством обработки *XML* -документов. В данной лекции мы будем рассматривать именно его.

#### **Установка расширения DOM XML**

Для того чтобы установить расширение *DOM XML*, нужно сделать следующее.

* В *файле настроек* PHP (*php.ini*) раскомментировать строку, касающуюся этого расширения ( extension=php\_domxml.dll для Windows, либо extension=php\_domxml.so для Linux-платформ).
* Скопировать файл расширения ( php\_domxml.dll или php\_domxml.so ) в папку, где находятся расширения ( *extension\_dir*).
* Подключить расширение iconv так же, как в пунктах выше (иногда это расширение устанавливается автоматически вместе с domxml ).
* Скопировать дополнительные библиотеки в системную папку system (Windows 98) или system32 (WindowsNT/2000/XP). В первую очередь это библиотеки libxml2 и iconv, затем libxslt, libexslt и zlib.
* Перезапустить сервер.

Следует проверить, правильно ли установлена переменная *extension\_dir* в файле настройки *php.ini*. Если она не указывает на директорию, где находятся библиотеки расширений PHP, то ни одно из расширений подключить не удастся.

Чтобы проверить, установилось ли расширение, можно создать простейший скрипт, который будет выводить все настройки PHP-интерпретатора (это делает функция *phpinfo()* ). Другой вариант – попробовать использовать какую-нибудь функцию из данного расширения. Например, можно попробовать получить версию используемой библиотеки libxml с помощью функции *domxml\_version()*. Но этот способ не очень хорош, поскольку расширение экспериментальное (это значит, что некоторые функции в каких-то определенных условиях могут и не работать), да и функции еще надо изучить, прежде чем их использовать.

<?

// выводит информацию о настройках PHP

phpinfo();

// отображает используемую версию

// библиотеки libxml

echo domxml\_version();

?>

Пример 14.1. Проверка, установлено ли расширение DOM XML

### **Взаимодействие PHP и XML посредством DOM XML**

Что происходит, если взаимодействие *PHP* и *XML* осуществляется с помощью *объектной модели* стандарта *DOM*? *Модуль* *DOM XML*определяет в *PHP* несколько классов, таких как *DomNode*, *DomDocument*, *DomElement*, *DomText* и *DomAttribute*, большинство из которых идут из ядра стандарта *DOM*. Почти для всех классов (в частности, для перечисленных выше) *класс* *DomNode* является *родительским*, поэтому его свойства и методы наследуются всеми остальными классами.

Если рассмотреть произвольный *XML* -документ, то классу *DomDocument* будет соответствовать сам этот документ, классу *DomElement*– каждый *XML* -тег, классу *DomAttribute* – атрибуты тегов, а классу *DomText* – содержание XML- *элементов*. В то же время классу *DomNode* будет соответствовать каждый из перечисленных *элементов**XML* -документа.

Рассмотрим коллекцию, содержащую описания персон. Если каждую из них мы описываем с помощью таких характеристик, как фамилия, имя, дата рождения и электронный *адрес*, то *структура* коллекции " Личности ", где хранится *информация* обо всех известных нам персонах, может быть представлена следующим образом.

<?xml version="1.0"?>

<collection>

<person id="10">

<name>

<first>Nick</first>

<last>Petrov</last>

</name>

<birth>

<day>23</day>

<month>12</month>

<year>89</year>

</birth>

<email> nick@ngs.ru

</email>

</person>

<person id="20">

<name>

<first>Bob</first>

<last>Ivanov</last>

</name>

<birth>

<day>03</day>

<month>05</month>

<year>90</year>

</birth>

<email> bob@ngs.ru

</email>

</person>

</collection>

Пример 14.2. Коллекция Личности в виде XML-файла (persons.xml)

В дальнейшем, приводя примеры, мы будем использовать этот *файл*.

Нам необходимо научиться читать, добавлять, изменять и искать информацию, находящуюся в *XML* -файлах.

#### **Перевод данных XML-файла в объекты и классы PHP**

Первое, что нужно сделать, если мы хотим работать с *XML* -данными в PHP при помощи расширения *DOM XML*, это перевести имеющиеся данные в объекты и классы *DOM*. Это можно сделать несколькими способами.

1. С помощью функции *domxml\_open\_mem* .

Синтаксис:

object domxml\_open\_mem (string str)

В качестве параметра эта функция принимает строку str, содержащую *XML* -документ. Результатом ее работы является объект класса, называемого *DOMDocument*.

1. С помощью функции *domxml\_open\_file* .

Синтаксис:

object domxml\_open\_file (string filename)

Эта функция обрабатывает *XML* -файл, имя которого задается параметром filename, и переводит его в объект класса *DOMDocument*. Доступ к файлу производится только на чтение.

Такие функции, как *domxml\_open\_mem()* и *domxml\_open\_file()* , как правило, нужно вызывать перед вызовом любых других функций, связанных с расширением *DOM*.

Эти функции преобразуют *XML* -файл в дерево объектов. К таким объектам можно обращаться с помощью различных методов. В частности, для выделения *корневого элемента* используется метод *DomDocument->document\_element()* .

Еще существует функция *domxml\_new\_doc(string version)* , которая создает новый пустой *XML* -документ. Ее параметром является номер версии создаваемого документа. Но ее мы касаться не будем, а будем считать, что *XML* -файл уже создан.

<?

//считываем файл "persons.xml" в строку

$xmlstr = join('',file('persons.xml'));

// переводим строку с xml-файлом

// в дерево объектов. Если операция

// прошла неудачно, то выводим

// ошибку и прекращаем работу.

if(!$dom = domxml\_open\_mem($xmlstr)) {

echo "Ошибка при разборе документа\n";

exit;

}

// можно посмотреть, как выглядит

// этот объект

print\_r($dom);

echo "<hr>";

// выделяем корневой элемент

// дерева объектов.

// В нашем случае это будет

// элемент <collection>

$root = $dom->document\_element();

print\_r($root);

echo "<hr>";

?>

Пример 14.3. Перевод XML-файла в дерево объектов PHP и выделение корневого элемента

Итак, каждому *элементу* *XML* -файла мы поставили в соответствие какой-то объект. Теперь нужно научиться перемещаться по дереву объектов и обращаться с этими объектами: получать и изменять их значения, находить их *потомков* и предков, удалять объекты.

#### **Обход дерева объектов**

Для получения значения текущего узла (вне зависимости от его типа) используют метод *DomNode->node\_value()* или *DomNode->get\_content()* для получения содержимого узла.

Для получения значения атрибута используется метод *DomElement->get\_attribute (attr\_name)* . А метод *DomNode->child\_nodes()* возвращает массив *потомков* данного узла.

Для того чтобы сделать обход дерева объектов, полезно еще уметь различать объекты по типам, т.е. определять, является ли узел *элементом* (тегом), текстом, атрибутом и т.п. Для этого используются специальные константы. *XML\_ELEMENT\_NODE* определяет, является ли узел *элементом*, *XML\_ATTRIBUTE\_NODE* определяет, является ли узел атрибутом, и *XML\_TEXT\_NODE* определяет, является ли узел куском текста. Эти константы имеют целочисленные значения 1, 2 и 3 соответственно. Использование этих констант полезно, поскольку переводы строки, применяемые для *удобочитаемости* *XML* -файлов, тоже становятся узлами.

<?

// сначала делаем то же,

// что и в предыдущем примере

$xmlstr = join('',file('persons.xml'));

if(!$dom = domxml\_open\_mem($xmlstr)) {

echo "Ошибка при разборе документа\n";

exit;

}

$root = $dom->document\_element();

// Получаем массив потомков

// родительского узла

// (в нашем случае это массив <person>)

$nodes = $root->child\_nodes();

print\_r($nodes);

echo "<hr>";

// Начинаем обработку каждого

// узла в массиве

foreach($nodes as $node){

// Если текущий узел – один

// из узлов <person>, то

// продолжаем ее обработку,

// чтобы получить информацию

// об этой личности

if ($node->tagname=='person'){

// Создаем массив, куда

// будем собирать информацию

// о рассматриваемой личности

$currentPers = array();

// Получаем id личности,

// который хранится в атрибуте 'id'

$currentPers['id'] =

$node->get\_attribute('id');

// Получаем массив потомков

// <person>. Это вся

// информация о личности

// (<name>,<birth> и т.д.)

$persons\_info =

$node->child\_nodes();

// Перебираем все дочерние

// узлы $node

foreach ($persons\_info as $info){

// проверяем, является ли узел

// элементом (xml-тегом)

if ($info->type==

XML\_ELEMENT\_NODE) {

// тогда метод tagname

// возвратит имя этого

// элемента (тега), а метод

// get\_content() –

// его содержимое

$currentPers[$info->tagname] =

$info->get\_content();

}

}

// выводим на экран полученные

// массивы

print\_r ($currentPers);

echo "<br>";

}

}

?>

Пример 14.4. Обход дерева XML

Итак, мы научились обходить дерево *XML*. Теперь можно попытаться что-нибудь найти в *XML* -файле. Правда, делать это не совсем удобно опять же из-за переносов строк, которые мы использовали при написании *XML* -файла. Пусть наш *XML* -файл записан в строку, а точнее, в нем есть следующая строка:

...

<person id="20">

<name>

<first>Иван</first>

<last>Иванов</last>

</name>

...

Тогда в наш предыдущий пример вставим (после вывода на экран полученных массивов) строчку для поиска электронного адреса Ивана Иванова.

...

$str = $currentPers["email"];

if ($currentPers["name"] ==

"Иван Иванов" )

echo "Здравствуйте, Иван! " .

"Ваш e-mail $str";

...

#### **Добавление новых элементов в XML-документ**

Далее разберем задачу, как можно добавить в нашу базу данных новую личность средствами php.

Сначала нужно скопировать описание личности (считаем, что все личности описываются с помощью стандартного набора характеристик, как в файле persons.xml ). Это делается с помощью метода *DomNode->clone\_node()* . Таким образом, мы клонируем *элемент* <person> и все его внутренние *элементы* (содержание тегов не копируется).

Потом можно установить какие-нибудь значения для *элементов* описания личности. Например, задать имя человека, дату его рождения и т.п. В конце нужно записать полученное описание личности в качестве *потомка**корневого элемента* в дерево *DOM* с помощью метода *DomNode->append\_child(new\_node)* , где в качестве параметра передается созданный объект (новый узел).

В PHP до версии 4.3 перед добавлением *потомка* к узлу с помощью данной функции этот *потомок* сначала копировался. Таким образом, новый узел являлся новой копией, которая могла изменяться без изменения узла, переданного как параметр в эту функцию. В более поздних версиях PHP новый узел удаляется из существующего контекста, если он уже есть в дереве. Такое поведение соответствует спецификациям W3C.

Для *удаления узла* можно воспользоваться методом, применив его к узлу, который требуется удалить, т.е. *DomNode->unlink\_node()*.

// Для того чтобы добавить описание

// новой личности, нужно знать,

// как описывается каждая личность.

// Выбираем элемент person,

// который содержит описание личности

$elements = $dom->get\_elements\_by\_tagname("person");

$element = $elements[0];

//вычисляем родителя и потомков

$parent = $element->parent\_node();

$children = $element->child\_nodes();

// клонируем элемент person

$person = $element->clone\_node();

// устанавливаем новой

// личности идентификатор

$attr = $person->set\_attribute("id", "30");

// если у личности были потомки,

// то их тоже надо клонировать

foreach ($children as $child){

//клонируем ребенка

$node = $child->clone\_node();

//получаем массив внуков

$grand\_children = $child->child\_nodes();

// если ребенок имеет потомков,

//т.е. массив внуков не пуст, то

if (count($grand\_children)<>1){

//клонируем каждого внука

//и присоединяем к уже

//клонированному ребенку

foreach($grand\_children as $grand\_child){

$lastnode = $grand\_child->clone\_node();

//записываем в нужные теги

//подходящие значения

if ($grand\_child->tagname=="first")

$cont = $lastnode->set\_content("Nina");

if ($grand\_child->tagname=="last")

$cont = $lastnode->set\_content("Saveljeva");

if ($grand\_child->tagname=="day")

$cont = $lastnode->set\_content("7");

if ($grand\_child->tagname=="month")

$cont = $lastnode->set\_content("06");

if ($grand\_child->tagname=="year")

$cont = $lastnode->set\_content("1981");

$newlastnode = $node->append\_child($lastnode);

}

}

if ($child->tagname=="email") {

$cont = $node->set\_content("help@intuit.ru");

}

$newnode2 = $person->append\_child($node);

}

$newnode = $parent->append\_child($person);

//dump\_mem создает XML-документ из dom

//представления

echo "<PRE>";

$xmlfile = $dom->dump\_mem(true);

// посмотрим в браузере,

// что получилось

echo htmlentities($xmlfile);

echo "</PRE>";

// запишем полученный XML-файл

// в файл "test.xml"

$h = fopen("test.xml","a");

if (!fwrite($h, $xmlfile)) {

print "Cannot write " . "to file ($filename)";

exit;

}

}

Пример 14.5. Добавление описания новой личности в каталог

### ***Заключение***

Итак, мы изучили ряд функций, позволяющих манипулировать данными, хранящимися в *XML* -формате. Это, конечно же, далеко не полный перечень существующих функций. В версии *PHP5* он значительно усовершенствован и в большей степени соответствует стандарту *DOM*. Тем не менее *знание* приведенных здесь основных функций может оказаться полезным при решении конкретных прикладных задач.