Обход дерева DOM

Извлечение данных из XML-документа, его модификация и даже создание — все эти операции сопряжены с обходом описывающего документ дерева DOM. Осуществить этот обход довольно несложно, особенно, если структура дерева заранее известна. Для этого предназначен ряд свойств и методов класса XMLNode, которые мы подробно обсудим в этом разделе.

Начинается любой обход с получения ссылки на объект в дереве DOM, описывающий корневой тег документа. Эту ссылку хранит объект класса XML, через который текст документа был передан анализатору. Все три способа получить ее приведены ниже:

```
var xml_doc:XML=new XML("<text>Привет</text>");

var root_node_1:XMLNode=xml_doc.firstChild;

var root_node_2:XMLNode=xml_doc.lastChild;

var root_node_3:XMLNode=xml_doc.childNodes[0];

trace(root_node_1==root_node_2 && root_node_1==root_node_3 ); // Выводит: true
```

Все способы получения доступа к объекту корневого узла основаны на том, что объект класса XML формально считается его родительским узлом. Поэтому можно применять те же средства, что используются при обходе дерева DOM для перехода от родительского узла к дочернему.

Успешно перейдя к объекту корневого узла документа (или к объекту любого другого

узла), в первую очередь нужно проверить, есть ли у него дочерние узлы. Сделать это позволяет метод hasChildNodes() класса XML, который возвращает true, если они есть, и false, если узел пустой.

Пример:

```
var xml_doc:XML=new XML("<text>Привет</text>");

trace(xml_doc.firstChild.hasChildNodes()); // Выводит: true
```

Перед тем, как перейти к исследованию дочерних узлов корневого узла, необходимо считать данные из его атрибутов. Хранятся атрибуты в виде свойств особого объекта, на который указывает свойство attributes объекта узла. Имена у этих свойств будут такие же, как и атрибутов, на основании которых они были созданы. И в этом кроется потенциальная опасность. Дело в том, что синтаксис имен XML менее строг, чем ActionScript. Например, в имя может входить двоеточие. Однако в ActionScript такой идентификатор вызовет сбой при компиляции. Чтобы этого не произошло, обращаться к описывающим атрибуты свойствам следует не через оператор “.”, а посредством оператора “[]”.

Значения у свойств объекта attributes всегда имеют строковый тип.

```
var xml_doc:XML=new XML("<text id='#1083'>Привет</text>");

trace(xml_doc.firstChild.attributes["id"]); // Выводит: #1083
```

Если у объекта корневого узла есть дочерние узлы, их нужно исследовать. Для этого их необходимо последовательно перебрать. Имеется два способа это сделать:

• У класса XMLNode имеется свойство childNodes, которое указывает на массив, хранящий ссылки на объекты всех дочерних узлов данного узла. Расположены ссылки в том же порядке, в котором теги дочерних узлов прописаны в теге родительского узла. Следовательно, чтобы исследовать все дочерние узлы, следует запустить цикл и перебрать все элементы массива childNodes:

```
var xml_doc:XML = new XML("<text><t1/><t2/><t3/></text>");

var ch_nodes:Array = xml_doc.childNodes[0].childNodes;

for (var i = 0; i<ch_nodes.length; i++) {

trace(ch_nodes[i].nodeName); // Выводит: t1, t2, t3

}
```

• Если обработка XML-документа осуществляется при помощи рекурсивных функций, то применение массива childNodes не всегда удобно. Эффективный и компактный рекурсивный алгоритм можно организовать при условии, что от одного дочернего узла можно перейти к другому, не обращаясь к родительскому узлу. В DOM, используемой ActionScript, это вполне реально благодаря наличию свойств previousSibling, nextSibling, а также firstNode и lastNode.

В дереве DOM все объекты, описывающие дочерние узлы некоторого узла, связаны между собой. Каждый объект “знает”, какой узел прописан после соответствующего ему узла, а какой — до. Ссылку на объект следующего за данным узлом узла можно получить через свойство nextSibling. Ссылку на объект, описывающий узел, предшествующий данному, хранит свойство previousSimbling.

Чтобы начать обход дочерних узлов посредством свойства nextSimbling или

previousSimbling, нужно иметь ссылку на объект первого или последнего узла. Получить ее можно при помощи свойства firstChild (первый узел) или lastChild (последний узел). В качестве примера использования рекурсивного обхода дерева DOM приведем функцию obhod(), подсчитывающую количество узлов в документе. Ее идея проста. Она получает ссылку на узел и увеличивает значение переменной-счетчика на 1. Затем она проверяет, есть ли у узла дочерние узлы. Если они есть, то создается еще одна активация функции и ей передается ссылка на первый дочерний узел. Далее функция проверяет, имеется ли узел, который был бы прописан после данного. Если он обнаруживается, то ссылка на него передается новой активации obhod():

```
var xml_doc:XML = new XML("<text><t1/><data>Привет</data><t3/></text>");

var col:Number = 0; // Переменная-счетчик

function obhod(node:XMLNode):Void {

col++;

if (node.hasChildNodes()) {

obhod(node.firstChild);

}

if (node.nextSibling != null) {

obhod(node.nextSibling);
```

```
}

}

obhod(xml_doc.firstChild); // Запускаем обход

trace(col); // Выводит: 5
```

Рекурсия — это чрезвычайно мощный инструмент для обработки XML. Ее нужно использовать, когда структура обрабатываемого документа неизвестна. При этом возникает задача обхода дерева неизвестной степени вложенности, которую довольно проблематично решить без рекурсии. Если же структура документа проста и заранее известна, то для извлечения из него данных лучше ограничиться обычными циклами. Самым важным ограничителем в применении рекурсии для обработки XML-документов является то, что в стеке не может находиться более 255 активаций. Впрочем, это ограничение можно преодолеть, изменив тег scriptLimits SWF-файла при помощи утилит Flexm или SWF ScriptLimits Injector. При необходимости, можно перейти не только от родительского узла к дочернему, но и наоборот, от дочернего к родительскому. У любого объекта класса XMLNode есть свойство parentNode, которое хранит ссылку на родительский узел. Пример его использования:

```
var xml_doc:XML = new XML("<text>Привет</text>");

trace(xml_doc.firstChild.firstChild.parentNode.nodeName); // Выводит: text
```

После того, как ссылка на объект узла будет получена, нужно определить его тип. Как вы помните, в DOM, используемой ActionScript, реализовано всего два типа узлов: ELEMENT и TEXT.

Узнать, является ли узел обычным или текстовым, позволяет свойство nodeType. Оно равняется 1, если узел “обычный”, и 3, если узел текстовый. Пример:

```
var xml_doc:XML = new XML("<text>Привет</text>");

trace(xml_doc.firstChild.firstChild.nodeType); // Выводит: 3
```

На первый взгляд значения, которые может принимать свойство nodeType, кажутся очень странными. Почему 1 и 3? Почему не 0 и 1 или 1 и 2? Дело в том, что в “настоящей” DOM, разработанной комитетом W3C, типов узлов не 2, а 12. Однако анализатор XML Flash-плеера поддерживает только два из них. Для большего же соответствия стандарту, номера для типов поддерживаемых узлов nodeType возвращает такие, какие им соответствуют в DOM от W3C. Если узел относится к типу ELEMENT, то, чтобы понять, какие данные в нем содержаться, следует прочитать его имя. Служит для этого свойство nodeName класса XMLNode. Имя узла оно возвращает в форме строки. Если узел является текстовым, то свойство nodeName будет равняться null.

```
var xml_doc:XML = new XML("<text></text>");

trace(xml_doc.firstChild.nodeName); // Выводит: text
```

Если узел является текстовым, то единственная операция, которая с ним должна быть проведена — это считывание содержащегося в нем текста. Выполнить ее позволяет свойство nodeValue класса XMLNode. Если узел относится к типу ELEMENT, а не TEXT, то свойство nodeValue будет равняться null.

Пример:

```
var xml_doc:XML = new XML("<text>Привет</text>");
```

trace(xml_doc.firstChild.firstChild.nodeValue); // Выводит: Привет

Официально класс XMLNode не поддерживает пространств имен XML. Однако если снять защиту от перечисления циклом for-in с прототипа этого класса, можно обнаружить 3 свойства и 2 метода, предназначенных, как следует из их названия, для работы с пространствами имен. Немного поэкспериментировав, несложно понять назначение каждого из этих недокументированных элементов. Кратко опишем их (для более полного описания одних лишь экспериментов недостаточно):

• namespaceURI. Свойство хранит URI пространства имен, к которому относится узел. Как вы помните, само URI прописывается в атрибуте xmlns тега, который является родительским по отношению к данному.

• prefix. Свойство возвращает строку с префиксом пространства имен, используемую в имени узла.

• localName. Свойство хранит имя узла без префикса (свойство nodeName возвращает имя вместе с префиксом).

• getPrefixForNamespace(). Метод определяет префикс пространства имен по его URI.

• getNamespaceForPrefix(). Метод определяет URI пространства имен по префиксу. Так как описанные методы и свойства являются недокументированными, компилятор о них не знает. Поэтому обращаться к ним следует не через оператор точки, а посредством оператора “[]”.

Пример:

```
var xml_text:String="<text xmlns:new='http://www.mysite.ru'><new:node/> </text>"

var xml_obj:XML=new XML(xml_text);

var ch_node:XMLNode= xml_obj.firstChild.firstChild;

trace(ch_node["namespaceURI"]); // Выводит: http://www.mysite.ru

trace(ch_node["prefix"]); // Выводит: new

trace(ch_node.nodeName); // Выводит: new:node

trace(ch_node["localName"]); // Выводит: node

trace(ch_node["getPrefixForNamespace"]("http://www.mysite.ru")); // Выводит: new

trace(ch_node["getNamespaceForPrefix"]("new")); // Выводит: http://www.mysite.ru
```

Создание и модификация XML-документа

Чтобы создать при помощи ActionScript XML-документ, нужно сформировать описывающее его дерево DOM. Эта задача решается в несколько этапов. На первом этапе следует изготовить объект класса XML, который послужит фундаментом для дерева DOM:

```
var doc:XML=new XML();
```

Теперь можно начать формирование дерева DOM. Для этого нужно создать объекты класса XMLNode и связать их в нужном порядке. Имеется три способа изготовить описывающий узел объект:

• Если нужно создать узел типа Element, то следует использовать метод createElement() класса XML. В качестве параметра данный метод принимает строку с именем узла. Обратите внимание, что на момент создания узел не связан с каким-либо родительским узлом и не имеет дочерних узлов. Как его встроить в дерево DOM, мы покажем чуть ниже.

Пример:

```
var doc:XML=new XML();

var new_node:XMLNode=doc.createElement("data");
```

• Чтобы изготовить объект, описывающий текстовый узел, следует обратиться к методу createTextNode() класса XML. Текст узла передается методу в параметре в виде строки.

```
var doc:XML=new XML();
```

var new_text_node:XMLNode=doc.createTextNode("Привет");

• Довольно часто XML-документы содержат несколько экземпляров одного узла, но с разными значениями атрибутов или различными дочерними узлами. В этом случае создавать каждый узел индивидуально не технично. Проще задать лишь один узел, а затем просто “размножить” его копированием. Служит для этого метод cloneNode() класса XMLNode.

У метода cloneNode() имеется два режима работы, определяемых значением параметра. В первом режиме (параметр равняется true) узел копируется со всеми дочерними узлами. Во втором режиме (параметр равняется false) клонируется лишь сам узел, без дочерних узлов. Атрибуты узла копируются в обоих режимах.

var doc:XML=new XML("<doc><text>Привет</text></doc>");

var cloned_node:XMLNode=doc.childNodes[0].childNodes[0].cloneNode(true);

trace(cloned_node); // Выводит: <text>Привет</text>

Если у узла должны быть атрибуты, соответствующие свойства должны быть созданы у объекта, на который указывает свойство attributes:

var doc:XML=new XML();

var new_node:XMLNode=doc.createElement("text");

new_node.attributes.author="Pushkin", new_node.attributes.title="Onegin";

Создав объект узла, его нужно встроить в дерево документа. Служат для этого методы appendChild() и insertBefore() класса XMLNode. Метод appendChild() используется, если новый узел должен быть встроен в качестве последнего дочернего узла того узла, через описывающий который объект был вызван данный метод. Также appendChild() применяется, если последовательность расположения дочерних узлов не имеет значения. В качестве параметра метод appendChild() принимает описывающий узел объект класса XMLNode.

Пример:

```
var doc:XML = new XML();

var node_1:XMLNode = doc.createElement("doc");

var node_2:XMLNode = doc.createElement("text");

var text_node_3:XMLNode = doc.createTextNode("Привет");

node_2.appendChild(text_node_3);

node_1.appendChild(node_2);

doc.appendChild(node_1);

trace(doc); // Выводит: <doc><text>Привет</text></doc>
```

Обратите внимание, что для того, чтобы создать корневой узел дерева DOM, метод appendChild() должен быть применен к объекту класса XML. При формировании же структуры дерева данный метод вызывается соответствующими узлам объектами класса XMLNode.

Если последовательность расположения дочерних узлов важна, то вместо метода appendChild() следует использовать метод insertBefore(newChild, beforeChild). Здесь newChild — описывающий узел объект класса XMLNode, beforeChild — ссылка на объект дочернего узла, перед которым должен быть вставлен данный узел.

Пример:

```
var doc:XML = new XML("<doc><tag1/><tag2/><tag3/></doc>");

var new_child:XMLNode=doc.createElement("tag4");

doc.firstChild.insertBefore(new_child, doc.firstChild.childNodes[2]);

trace(doc); // Выводит: <doc><tag1/><tag2/><tag4/><tag3/></doc>
```

При необходимости, к XML-документу можно добавить XML-объявление или DTD-объявление. Служат для этого рассмотренные ранее свойства docTypeDecl и xmlDecl.

Обмен XML-данными с сервером

XML-документ — это просто текстовый файл. Поэтому нет принципиальных отличий между передачей на сервер данных (или получением их с него) в формате пар

имя-значение и XML. Для этого используются уже хорошо знакомые нам по классу LoadVars (он был описан в прошлой статье) элементы: методы send(), load(), sendAndLoad(), getBytesTotal(), getBytesLoaded(), addRequestHeader(); свойства contentType, loaded; события onData и onLoad. Ввиду того, что эти элементы мы изучили весьма подробно, а также того, что между их реализациями в классах LoadVars и XML нет существенных различий, повторно рассматривать их мы не будем. Мы ограничимся приведением лишь некоторых неочевидных фактов, связанных с пересылкой XML-данных:

• Для отправки XML-документа на сервер стоит использовать HTTP-метод POST, но ни в коем случае не GET. Дело в том, что длина URL зачастую ограничена 256 символами, поэтому документ может просто “не влезть”. Однако нужно помнить, что применить метод POST можно лишь в том случае, если фильм воспроизводится в браузере. Если же он проигрывается автономным плеером, то метод POST будет недоступен.

• Загрузив XML-документ, Flash-плеер пытается произвести его разбор. По завершении этой попытки возникает событие onLoad. Однако имеется возможность работы с XML-текстом напрямую. Для этого нужно обратиться к событию onData. Данное событие возникает непосредственно после того, как заканчивается загрузка документа. Текст документа передается обработчику onData в качестве параметра. Если в коде есть обработчик события onData, то преобразования XML-документа в дерево объектов DOM не производится.

• По умолчанию в качестве заголовка типа содержимого Flash-плеер использует application/xwww- form-urlencoded. Это не всегда приемлемо. Если вы используете серверную технологию, поддерживающую разбор XML-данных, заголовок по умолчанию следует заменить строкой text/xml. Для этого следует переопределить свойство contentType.

• Если XML-данные передаются на сервер не в UTF-8, информацию об этом обязательно следует включить в XML-объявление при помощи атрибута encoding.

Для оперативного обмена XML-данными с сервером нужно использовать класс XMLSocket. Данный класс отлично приспособлен для выполнения работы такого рода. Любой полученный текст передается XML-анализатору по умолчанию. Если преобразование в дерево DOM происходит успешно, то возникает событие onXML,

обработчику которого передается полученный объект класса XML.