Структура документа в DOM

Структура документа в DOM

DOM представляет XML-документ как иерархию объектов нового типа - узлов (node). На вершине иерархии находится узел (объект)document, который представляет весь документ. В качестве узлов в DOM представлено все содержание документа: XML элементы, атрибуты этих элементов и текст XML элементов-контейнеров.

Пример HTML документа

Рассмотрим пример небольшого фрагмента HTML документа:

<HTML>

<HEAD>

<TITLE>

Overview of the DOM

</TITLE>

</HEAD>

<BODY>

<H1>

Иерархия узлов

</H1>

<P>

На вершине иерархии находится узел

<TT>

Document

</TT>

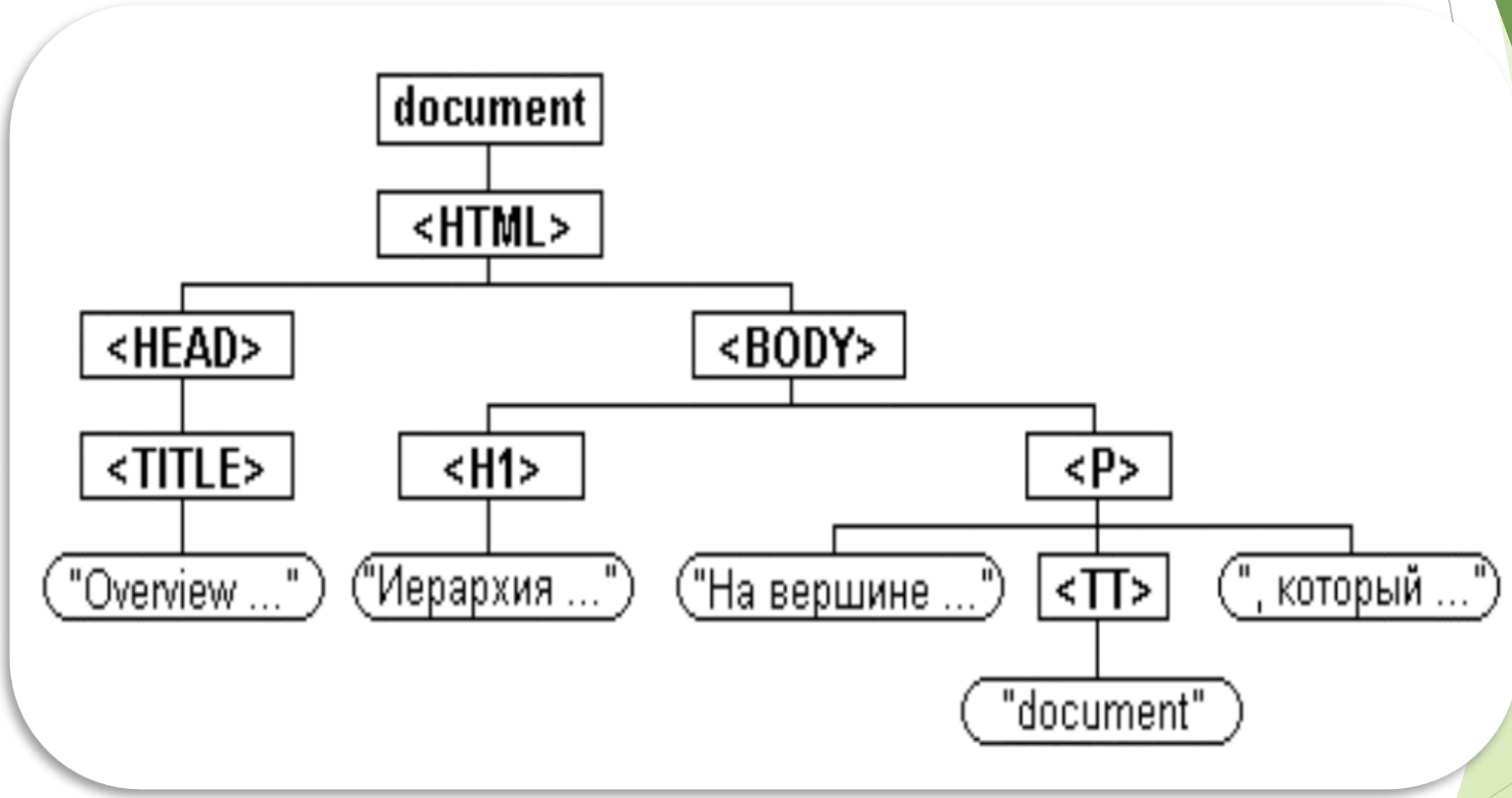
, который представляет в DOM сам документ.

</P>

</BODY>

</HTML>

Структура документа



Структура документа
Продолжение

Элементы, имеющие открывающий и закрывающий тэги (элементы-контейнеры), могут иметь потомков. Текст, атрибуты и элементы типа IMG, не имеющие закрывающего тэга, иметь потомков не могут. Также узлы типа атрибут могут быть потомками только узлов типа элемент. В HTML разные элементы, например, P или IMG, имеют разный набор допустимых для них атрибутов. В DOM узлы-элементы разного вида также имеют разный набор допустимых для них узлов-атрибутов. За редким исключением они соответствуют HTML4. Списки рекомендованных атрибутов для различных элементов можно найти в документации W3C.

Узлы разного типа (document, элементы, атрибуты и текст) имеют свой набор свойств и методов, которые позволяют через сценарии манипулировать ими. Ряд узлов-элементов (объектов) имеют свои собственные свойства и методы. Так, например, объект элемента TABLE имеет метод createTHead.

Пример документа

При рассмотрении свойств и методов узлов будем использовать следующий фрагмент документа:

```
<DIV>
<UL ID="components">
<LI>
HTML
</LI>
<LI>
CSS
</LI>
<LI>
Javascript
</LI>
</UL>
</DIV>
```

В DOM этому фрагменту соответствует ветка дерева, растущая из узла <DIV> к узлу <UL>. Здесь она разветвляется на три веточки по числу узлов <LI> (узлы-атрибуты не принято включать в состав дерева). Каждый из этих узлов имеет
по одному побегу, который заканчивается текстовым узлом.

Навигация по дереву документа

Навигацию по дереву документа можно начинать с любого узла-элемента, для которого мы знаем идентификатор, присваиваемый ему в качестве значения атрибута ID. Ссылку на такой узел можно получить с помощью метода getElementById() объекта document. Параметром метода является идентификатор. Следующая строка кода присваивает переменной oList ссылку на наш список:

```
var oList =  document.getElementById("components")
```

Навигация по дереву документа

Стартуя с некоторого узла, мы можем бродить по дереву в любом направлении, используя ряд

свойств узлов. Так, узлы-элементы и текстовые узлы имеют свойство parentNode, которое возвращает ссылку на родительский узел.

Возьмем для примера узел (объект) oList. Ссылку на родительский элемент DIV этого узла можно получить следующий образом:

```
var oParent = oList.parentNode
```

Навигация по дереву документа

Узлы-элементы и текстовые узлы, являющиеся

дочерними некоторого узла, входят в состав коллекции childNodes этого узла.

Узлы-атрибуты составляют отдельную коллекцию attributes. К каждому из них

можно обращаться по индексу массива. Например,

строка кода

var oItem1 = oList.childNodes[1]

присваивает переменной oItem1 ссылку на элемент

<LI> CSS </LI> нашего списка. Именно этот элемент

представлен в DOM как узел childNodes[1] узла oList.

Навигация по дереву документа

Первый (childNodes[0]) и последний элементы коллекции имеют специальные имена: firstChild и lastChild. Эти имена являются свойствами родительского узла.

Каждый из элементов коллекции имеет свойства previousSibling и

nextSibling. Эти свойства хранят ссылку на ближайших братков элемента -

предыдущий и последующий элементы коллекции (возвращают null, когда братков нет). Так, элемент childNodes[1] является свойством

nextSibling элемента childNodes[0] и свойством previousSibling элемента

childNodes[2].

Навигация по дереву документа

Используя эти свойства, мы можем получить ссылку на узел

childNodes[1] любым из следующих способов:

oList

.firstChild.nextSiblingoList

.childNodes[2].previousSibling

Ссылка на более удаленные узлы как по горизонтали, так и по вертикали дерева формируется путем слияния ссылок на ближайших родственников по стандартным правилам объектно-ориентированного программирования.

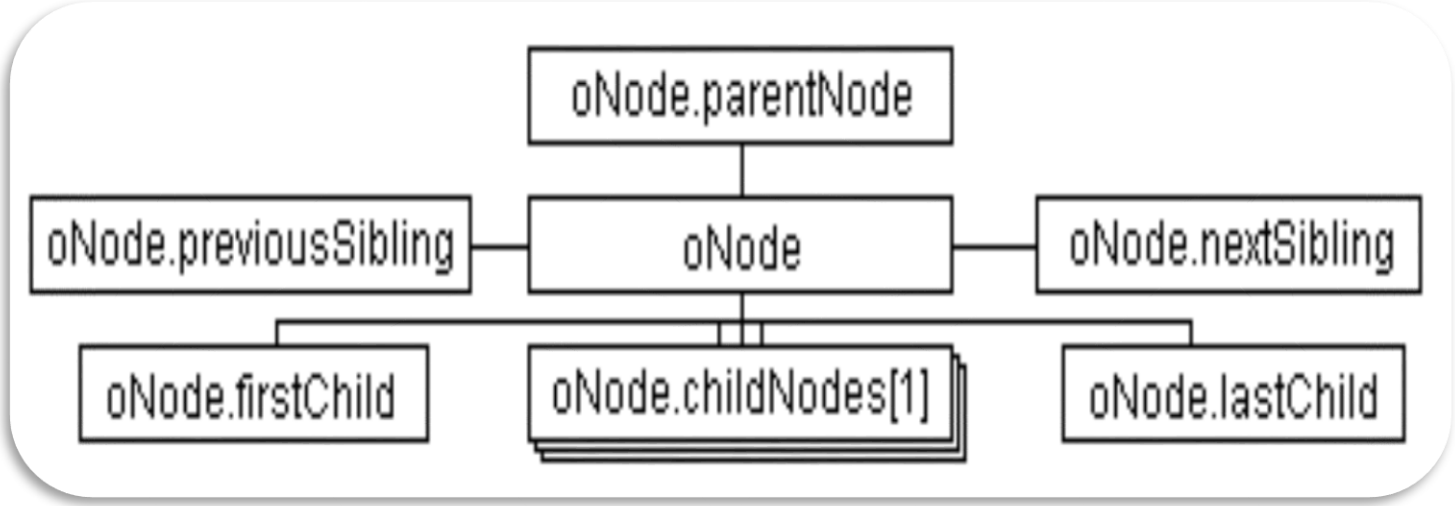
Т ак,

oList.childNodes[1].firstChild является ссылкой на текст"CSS"элемента

<LI> CSS </LI>

нашего списка.

Схема узлов документа



Все описанные выше свойства узлов (parentNode, firstChild, lastChild, nextSibling и previousSibling), необходимые для навигации по дереву документа, являются свойствами только для чтения.

Свойства-характеристики узлов

Свойство узла nodeType (только для чтения) возвращает 1, 2 или 3 для узлов, соответствующих тэгу, атрибуту или тексту, соответственно. Свойство nodeName (только для чтения) возвращает имя HTML тэга, которому соответствует данный узел, например, P для параграфа или UL для ненумерованного списка.

Для узлов-атрибутов nodeName возвращает название атрибута, а для тестовых узлов возвращает #text.

Текстовые узлы имеют еще одно очень важное свойство: nodeValue. Это свойство для чтения и записи хранит содержание текстового узла. Для элементов оно возвращает null, а для атрибутов - значение атрибута.

Свойства-характеристики узлов

Свойство узла nodeType (только для чтения) возвращает 1, 2 или 3 для узлов, соответствующих тэгу, атрибуту или тексту, соответственно. Свойство nodeName (только для чтения) возвращает имя HTML тэга, которому соответствует данный узел, например, P для параграфа или UL для ненумерованного списка.

Для узлов-атрибутов nodeName возвращает название атрибута, а для тестовых узлов возвращает #text.

Текстовые узлы имеют еще одно очень важное свойство: nodeValue. Это свойство для чтения и записи хранит содержание текстового узла. Для элементов оно возвращает null, а для атрибутов - значение атрибута.

Создание новых узлов

Метод createElement()

Метод createTextNode()

Создание новых узлов
Продолжение

Технику создания новых элементов обсудим на конкретном примере: мы хотим добавить к нашему списку элемент

<LI>

XML

</LI>

Этому HTML элементу в DOM соответствуют два узла: узел-элемент <LI> и текстовый узел "XML". Следовательно, нужно создать два новых узла. Новые узлы создаются с помощью методов createElement() и createTextNode() объекта document.

Создание новых узлов

Метод createElement()

Метод createElement() принимает в качестве параметра строку с названием тэга элемента и создает новый HTML элемент указанного типа. Например, строка кода

```
var oItem = document.createElement("LI")
```

создает новый элемент списка <LI>, у которого нет содержания. Метод возвращает ссылку на созданный им объект. Созданный выше элемент <LI> находится в памяти, но не входит в состав текущего документа. Для того, чтобы он стал частью документа, его надо добавить к существующим узлам документа с помощью методов insertBefore() или appendChild( ).

Создание новых узлов

Метод createTextNode()

Метод createTextNode() используется для создания текстового узла. Он принимает в качестве параметра строку текста, которая задает значение свойства nodeValue текстового узла. Например, строка кода

```
var oText = document.createTextNode("XML")
```

создает новый текстовый узел "XML". Метод возвращает ссылку на созданный им объект. Созданный текст еще не входит в состав текущего документа. Для того, чтобы он стал частью документа, его надо присоединить к существующим узлам документа с помощью методов appendChild(), replaceNode() или insertBefore().

Создание новых узлов
Метод createTextNode()

Если имеется ссылка на ненужный текстовый узел (например, oText), то можно не создавать нового узла, а воспользоваться уже существующим. Для этого надо просто присвоить новый текст в качестве значения свойства nodeValue существующего узла:

Text.nodeValue= "XML"

Итак, мы создали два новых узла: узел-элемент <LI> и текстовый узел "XML".

Редактирование дерева документа

DOM, как редактор, позволяет копировать, вставлять, замещать и удалять как отдельные узлы, так и целые ветви дерева документа.

Вспомним, что у нас есть два узла: узел-элемент <LI> и текстовый узел "XML". Оба узла находятся в памяти и мы хотим встроить их в текущий документ. Прежде всего, надо задать текст "XML" в качестве содержания элемента <LI>. Сделать это можно с помощью метода appendChild().

Редактирование дерева документа

Методы

Вставка: методы appendChild() и insertBefore()

Копирование: метод cloneNode()

Замещение: методы replaceChild()и replaceNode()*

Удаление: методы removeChild()и removeNode()*

Перестановка: метод swapNode()*

Дополнительные методы: applyElement()* и hasChildNodes()

Примечание. Методы, помеченные звездочкой*, не входят в список рекомендуемых W3C.

Редактирование дерева документа

Вставка: метод appendChild()

Данный метод добавляет элемент в конец коллекции childNodes узла, который его активизировал. Сам этот узел становится родительским узлом для узла, ссылку на который метод принимает в качестве параметра. Например, строка кода

oItem.appendChild(oText)

добавляет узел oText к узлу oItem. Отметим, что метод возвращает ссылку на объект, который он добавляет. В нашем случае это объект oItem.firstChild. Теперь мы имеет в памяти элемент (веточку дерева из двух узлов)

<LI> XML </LI>

Вставим эту веточку в текущий документ. Если мы хотим вставить ее в самый конец нашего списка, то надо, как и выше, использовать методappendChild():

oList.appendChild(oItem)

Редактирование дерева документа

Вставка: метод appendChild()

Поскольку узел oList, к которому мы присоединили узел oItem, является частью текущего документа, созданный нами элемент списка также становится частью документа. Теперь наш список выглядит так:

<UL ID="components">

<LI> HTML </LI>

<LI> CSS </LI>

<LI> Javascript </LI>

<LI> XML </LI>

</UL>

Вставить созданный нами элемент списка не в конец, а, скажем, после элемента списка<LI> HTML </LI> можно сделать с помощью метода insertBefore().

Редактирование дерева документа

Вставка: метод insertBefore()

В отличие от метода appendChild(), метод insertBefore() позволяет указать, в какое место коллекции childNodes будущего родительского узла будет вставлен новый узел. Как следует из названия, метод требует ссылки на узел, перед которым будет вставлен его новый браток. Мы создадим эту ссылку в отдельной строке, хотя это и необязательно. Итак, код

Var oBrother = oList.firstChild.nextSiblingoList.insertBefore(oItem, oBrother)

добавляет в коллекцию childNodes узла oList узел oItem сразу после узла childNodes[0]. В качестве первого параметра метод insertBefore() принимает ссылку на узел, который мы хотим добавить, а в качестве второго параметра – ссылку на узел, перед которым будет вставлен новый браток. Второй параметр метода является необязательным.

Редактирование дерева документа

Вставка: метод insertBefore()

Если родительский узел не имеет деток, то задавать его не следует. Если родительский узел имеет деток, а второй параметр не задан, то добавляемый узел становится самым последним среди деток родительского объекта. Метод insertBefore()возвращает ссылку на вставленный в документ объект.

Теперь первоначальный список выглядит так:

<UL ID="components">

<LI>HTML</LI>

<LI>XML</LI>

<LI>CSS</LI>

<LI>Javascript</LI>

</UL>

Редактирование дерева документа

Копирование: метод cloneNode()

Если необходимо скопировать некоторый узел вместе со всеми его атрибутами, то надо воспользоваться методом cloneNode(). В качестве параметра метод принимает выражение типа Boolean. Если оно равно false, то копируется только тот узел, который активизирует метод. Если параметр метода равен true, то копируется узел вместе со всеми его потомками. Например, строка кода

```
var oClone = oList.cloneNode(true)
```

копирует в память всю ветвь дерева, начинающуюся на узле oList, то есть весь наш список целиком. Метод возвращает ссылку на копию узла. Используя эту ссылку, мы можем в дальнейшем работать с этой копией, например, отредактировать ее и вставить в документ.

Редактирование дерева документа

Замещение: метод replaceChild()

Метод replaceChild() позволяет у узла, который его активизирует, заменить одного из его деток на нового. Ссылку на новый и на заменяемый узлы метод принимает в качестве первого и второго параметров, соответственно. Так, следующий фрагмент сценария

```
var oItem=document.createElement("LI")oItem.appendChild(document.createTextNode("JScript"))

oList.replaceChild(oItem, oList.lastChild)
```

создает сначала элемент списка с текстом "JScript", а затем заменяет им последний элемент нашего списка. Отметим, что метод возвращает ссылку на вставленный в документ узел.

Теперь список выглядит так:

```
<UL ID="components">
<LI>HTML</LI>
<LI>CSS</LI>
<LI>JScript</LI>
</UL>
```

Описанный выше пример надо рассматривать только как иллюстративный, поскольку тот же результат можно получить гораздо проще:

```
oList.lastChild.firstChild.nodeValue= "JScript"
```

Редактирование дерева документа

Замещение: метод replaceNode()

Если нужно заменить некоторый узел в документе другим узлом, то можно воспользоваться методом replaceNode(). Метод replaceNode() удаляет из документа узел, который его активизирует, и вставляет в документ вместо него новый узел, ссылку на который он принимает в качестве параметра. Заметим, что узел удаляется вместе со всеми своими атрибутами и потомками. Так, следующий фрагмент сценария

```
var oParagraph = document.createElement("P")
var oText = document.createTextNode("Составные части DHTML")
oParagraph.appendChild(oText)
oList.replaceNode(oParagraph)
```

создает сначала параграф с текстом "Составные части DHTML", а затем заменяет наш список oList на этот параграф. Отметим, что метод возвращает ссылку на вставленный в документ узел.

Метод replaceNode() не входит в список рекомендуемых W3C, но поддерживается Internet Explorer 5.

Редактирование дерева документа

Удаление: метод removeChild()

Метод removeChild() позволяет у узла, который его активизирует, удалить одного из его дочерних узлов. Ссылку на удаляемый узел метод принимает в качестве параметра. Например, строка кода

```
var oRemovedItem = oList.removeChild(oList.lastChild)
```

удаляет из нашего списка последний элемент. Метод возвращает ссылку на удаляемый им узел. Поскольку удаленный из документа узел остается в памяти, мы можем в дальнейшем работать с ним, используя эту ссылку. Например, пристроить в какой-нибудь другой список.

Редактирование дерева документа

Удаление: метод removeNode()

Если нужно удалить некоторый узел из документа, то можно воспользоваться методом removeNode(). В качестве параметра метод принимает выражение типа Boolean. Если оно равно false, то удаляется только тот узел, который активизировал метод. При этом, идущая от него ветвь дерева присоединяется к его родительскому узлу. Если параметр метода равен true, то узел удаляется вместе со всеми своими потомками. Например, строка кода

```
var oRemovedList = oList.removeNode(true)
```

удаляет из документа весь наш список целиком. Метод removeNode() возвращает ссылку на удаляемый им узел. Поскольку удаленный из документа узел остается в памяти, мы можем в дальнейшем работать с ним, используя эту ссылку.

Использовать false в качестве параметра надо осмысленно. Так, если мы применим метод removeNode() с параметром false к нашему списку, то его детки должны будут перейти к элементу DIV. Но, согласно требованиям HTML 4, элементы списка LI не могут размещаться в этом контейнере!

Подчеркнем, что этот метод не входит в список рекомендуемых W3C, но поддерживается Internet Explorer 5.

Редактирование дерева документа

Перестановка: метод swapNode()

Если необходимо переставить два узла в документе (в общем случае, две ветви дерева), то используется метод swapNode(). Данный метод меняет местами узел, который его активизировал, и узел, ссылку на который он принимает в качестве параметра. Например, строка кода

```
var oSwappedNode = oFirst.swapNode(oSecond)
```

переставляет узлы oFirst и oSecond и возвращает ссылку на узел, который активизирует метод.

Замечание: метод swapNode() не входит в список рекомендуемых W3C, но поддерживается Internet Explorer 5.

Редактирование дерева документа

Метод applyElement()

Описанные выше методы appendChild() и insertBefore() позволяют узлам заводить деток. Internet Explorer 5 поддерживает также метод, который позволяет узлу-элементу обзаводиться родителем. Таким методом является метод applyElement(), который принимает в качестве параметра ссылку на будущего родителя. Например, строка кода

var oParent = oChildNode.applyElement(oParentNode)

задает узел oParentNode в качестве родительского для узла oChildNode. При этом узел oParentNode лишается (если они у него были) как своего родителя, так и своих деток перед тем, как "заполучить" нового наследника. Метод возвращает ссылку на нового родителя.

Замечание: метод applyElement() не применим к текстовым узлам.

Редактирование дерева документа

Метод hasChildNodes()

Метод hasChildNodes() позволяет узнать есть или нет у узла потомки. Например, строка кода

oTestedNode.hasChildNodes()

возвращает true, когда узел oTestedNode имеет потомков, и false в противном случае.

Работа с атрибутами элементов

Метод createAttribute()

Метод setAttribute()

Метод removeAttribute()

Метод getAttribute()

Работа с атрибутами элемента
Продолжение

Все атрибуты узла-элемента (за исключением атрибута STYLE) составляют коллекцию attributes. W3C определяет эту коллекцию как массив с доступом по именам элементов. Например,

oNode.attributes.align

или

oNode.attributes["align"]

возвращает значение атрибута ALIGN узла oNode. (В документации Microsoft по DHTML сказано, что к элементам коллекции нужно обращаться по индексу массива. Но на практике работает описанное выше обращение по именам. Вероятно, документация обновляется реже, чем появляются новые версии броузера.)

Работа с атрибутами элемента
Продолжение

Имя атрибута надо набирать прописными буквами вне зависимости от того, в каком регистре они набраны в HTML-источнике. Свойство nodeName для узлов-атрибутов возвращает название атрибута, а nodeValue – значение атрибута. Свойство атрибутов specified позволяет узнать, определен или нет этот атрибут. Если, например, у узла oNode атрибут ALIGN определен, то

oNode.attributes["align"].specified

возвращает true, а если не определен, то false. (Такое поведение свойства specified реализовано и в Internet Explorer 5 вопреки документации Microsoft по DHTML).

Работа с атрибутами элементов
Метод createAttribute()

Метод createAttribute() объекта document позволяет создать узел-атрибут. В качестве параметра метод принимает имя атрибута.

Замечание: Internet Explorer 5 не поддерживает этот метод.

Работа с атрибутами элементов
Метод getAttribute()

Узнать текущее значение атрибута у элемента можно, активизируя метод getAttribute(), который требует в качестве параметра название этого атрибута.

Работа с атрибутами элементов

Метод getAttribute()

Атрибуты как новых элементов, так и тех, что заданы через HTML, можно задать либо традиционным способом, присваивая значение свойству (атрибуту) узла, либо с помощью метода setAttribute(). Оба способа демонстрируются ниже для атрибутаID. Так, любая из строк кода

oNode.id= "newItem"

oNode.setAttribute("id","newItem")

задает для элемента oNode в качестве идентификатора строку "newItem".

Метод setAttribute() требует два параметра. Первый параметр – строка, которая задает название атрибута. Второй параметр – строка, число или булево выражение, соответствующее значению атрибута. Согласно документации в Internet Explorer 5, по умолчанию названия атрибутов чувствительны к регистру, в котором они набраны. Если набор строчных и прописных букв не совпадает с использованным ранее в названии атрибута, то будет создан новый атрибут. Зависимость от регистра можно отменить, задав в качестве третьего параметра метода число 0.

Работа с атрибутами элементов

Метод removeAttribute()

Удалить атрибут у элемента можно, активизируя метод removeAttribute(), который требует в качестве параметра название этого атрибута. Если удаленный атрибут имеет значение по умолчанию, то оно восстанавливается. Согласно документации в Internet Explorer 5, по умолчанию в параметре набор строчных и прописных букв должен совпадать с использованным ранее в названии атрибута. Зависимость от регистра можно отменить, задав в качестве второго параметра метода число 0. Метод возвращает true при успешном выполнении действия и false в противном случае.

Дополнения, внесенные стандартом DOM Level 3

1.1 Переименование узла документа

В DOM Level 2 переименование узла было относительно «дорогой» операцией: необходимо было создать новый узел, скопировать в него все данные, вставить новый узел в дерево и удалить старый.

В DOM Level 3 в интерфейсе Document появился новый метод, который сделает все это за Вас: renameNode позволяет переименовать атрибут или элемент в дереве за один вызов.

Ниже приведен пример переименования узла с помощью средств DOM Level 3:

```
Element element = document.createElementNS
("http://example.com", "street");

 // если реализация позволяет переименовывать этот узел, он возвращается в тот же объект, где был создан изначально

element = document.renameNode(element, "http://example.com",
"address");
```

Дополнения, внесенные стандартом DOM Level 3

1.2 Перемещение узлов из одного документа в другой

Часто в памяти находятся два документа, и необходимо их слить или вставить часть одного документа в другой. В DOM Level 2 это можно сделать с помощью метода importNode интерфейса Document. Однако этот метод не изменяет исходное дерево. Вместо этого он создает клон исходного узла и его потомков, который Вы можете далее вставить в конечное дерево. Иногда это и надо, но в некоторых случаях необходимо только переместить узел из одного документа в другой. Это не только вынудит очистить исходные узлы, но еще и может оказаться «дорогим», если поддерево, которое перемещается, велико.

Используя DOM Level 3, можно это произвести с помощью метода adoptNode. Этот метод, также входящий в интерфейс Document, эффективно перемещает поддерево из одного документа в другой:

```
Node adoptedNode =document2.adoptNode(element);
```

Дополнения, внесенные стандартом DOM Level 3
2. Сравнение узлов

DOM Level 3 вносит набор методов, позволяющих сравнивать узлы разными способами. В него входят методы для проверки равенства двух узлов, а также того, как они расположены относительно друг друга в дереве документа.

Часто путают термины «идентичность» и «равенство». В языке Java идентичность проверяется с помощью оператора «==», а равенство с помощью метода, такого как equals. Два объекта будут идентичными, если они являются одним и тем же объектом в памяти. С другой стороны, для равенства объектов достаточно, чтобы они имели одинаковые характеристики.

DOM Level 3 предоставляет метод isEqualNode интерфейса Node для проверки двух узлов на равенство. Например, если Вы создадите 2 пустых элемента, имеющих имя «foo» и не имеющих атрибутов, они будут равны, но не идентичны. Для проверки идентичности двух узлов существует метод isSameNode.

Для проверки того, как узлы расположены относительно друг друга в дереве документа, используется метод compareDocumentPosition. Он может сообщить, является ли один узел предком или потомком другого, находится ли один узел перед другим или после другого и т.д.

Дополнения, внесенные стандартом DOM Level 3

3. Работа с текстом

В DOM Level 2 для того чтобы изменить текст одного узла-элемента, приходилось удалять его потомков, создавать текстовый узел с новым содержимым и вставлять его в качестве потомка узла-элемента. Приведем пример такой последовательности действий:

```
// предположим, что у элемента есть 2 потомка:
// комментарий и текстовый узел
NodeList list = elem.getChildNodes();
int len = list.getLength();
for (int i=0;i<len;i++)
{
elem.removeChild(list.item(i));
}
elem.appendChild(document.createTextNode("content"));
```

Дополнения, внесенные стандартом DOM Level 3

3. Работа с текстом (продолжение)

В DOM Level 3 это сделать гораздо проще. Новый атрибут для чтения/записи textContent позволяет с легкостью манипулировать текстовым содержимым. Установка этого атрибута удаляет все узлы-потомки и заменяет их одним текстовым узлом, если не установлено пустое значение. При чтении этот атрибут возвращает объединенное текстовое содержимое этого узла и его потомков.

Приведем пример получения текстового содержимого элемента и замена его на новое с использованием DOM Level 3:

```
String oldContent = elem.getTextContent();

elem.setTextContent("content");
```

Данные методы позволяют избежать манипуляции с узлами типа Text. Для того чтобы создать элемент, содержащий только текст, достаточно его создать и установить его textContent.

Другое полезное дополнение – атрибут wholeText интерфейса Text. Он возвращает весь текст, содержащийся в логически смежных текстовых узлах.